Огнёва М.В., Кудрина Е.В.

TURBO PASCAL: ПЕРВЫЕ ШАГИ

ПРИМЕРЫ И УПРАЖНЕНИЯ

«Научная книга»
2008

УДК 681.3.026(076.1)
ББК 32.973-01я73
О38

Огнева М.В., Кудрина Е.В.
О38 Turbo Pascal: первые шаги. Примеры и упражнения: Учеб. пособие. Изд.3-е, перераб. и доп. - Саратов: Изд-во "Научная книга", 2008. - 100 с.
ISBN 978-5-9758-0893-6

Данное пособие представляет собой учебно-методическую разработку по изучению основ программирования на языке Turbo Pascal. Каждый раздел данного пособия, кроме последнего, содержит: теоретический материал, примеры решения типовых задач, набор упражнений, предназначенных для закрепления материала и набор заданий, предназначенный для организации самостоятельной работы учащихся.

Пособие будет полезным школьникам старших классов, изучающим информатику на профильном уровне, абитуриентам, сдающим информатику в качестве вступительных испытаний, студентам начальных курсов для подготовки по дисциплинам компьютерного цикла и преподавателям всех ступеней образования для подготовки и проведения соответствующих занятий.

Рецензенты:

Федорова А.Г., кандидат физико-математических наук, доцент кафедры информатики и программирования, декан факультета компьютерных наук и информационных технологий Саратовского государственного университета им. Н.Г. Чернышевского

Кандратов Д. В., кандидат физико-математических наук, доцент кафедры математики и статистики, проректор по информатизации Поволжской академии государственной службы им. П.А. Столыпина

УДК 681.3.026(076.1)
ББК 32.973-01я73

Работа издана в авторской редакции

ISBN 978-5-9758-0893-6

ВВЕДЕНИЕ

В настоящее время в мире компьютеров существует множество языков программирования. Одну и ту же программу можно написать на Basic, Pascal, C. Какой из языков лучше? Ответ на данный вопрос не такой простой. Однако, можно с уверенностью сказать, что Pascal лучше других языков подходит для обучения основам программированию. И не удивительно, ведь этот язык был разработан швейцарским ученым Н.Виртом именно с этой целью.

Язык программирования Pascal оказался настолько удачным, что с момента его появления различными фирмами было создано большое количество компиляторов (компилятор – программа, переводящая инструкции языка программирования высокого уровня на язык инструкций процессора вычислительной машины). Одной из наиболее удачных версий стала разработка фирмы Borland, в которой в единое целое были объединены редактор текста и высокоэффективный компилятор. Созданная система получила название Turbo Pascal, а язык программирования, используемый в ней, стал называться Turbo Pascal (TP). Язык характеризуется расширенными возможностями по сравнению со стандартным языком Pascal, хорошо развитой библиотекой модулей, позволяющих использовать возможности операционной системы, создавать оверлейные структуры, организовывать ввод-вывод, формировать графические изображения и т.д. Среда программирования позволяет создавать тексты программ, компилировать их, находить и оперативно исправлять ошибки, компоновать программу из отдельных частей, включая стандартные модули, отлаживать и выполнять программу. К TP прилагается большой пакет справочной информации.

Пособие представляет собой вводный курс в язык программирования TP, поэтому начальных знаний по TP у читателей не требуется. Может быть рекомендовано в качестве пособия:

1) школьникам старших классов, изучающим информатику на профильном уровне;
2) абитуриентам, сдающим информатику в качестве вступительных испытаний;
3) студентам начальных курсов для подготовки по дисциплинам компьютерного цикла;
4) преподавателям всех ступеней образования для подготовки и проведения соответствующих занятий.

Пособие состоит из 10 разделов, в которых рассматриваются основные приемы программирования в среде TP, работа с различными типами данных, организация ввода-вывода. Каждый раздел пособия, кроме последнего, содержит:

1) теоретический материал;
2) примеры решения типовых задач;
3) набор упражнений, предназначенных для закрепления материала;
4) набор заданий, предназначенный для организации самостоятельной работы учащихся.

Предлагаемое пособие является третьим изданием, переработанным и дополненным. Основные изменения:

- объединены разделы «Цикл For» и «Реализация итерационных циклов»;

- объединены разделы «Одномерные массивы» и «Двумерные массивы»;
- добавлены разделы «Рекуррентные соотношения» и «Вычисление сумм и произведений»;
- в каждый раздел добавлены задания в пункт «Упражнения»;
- в разделы 1-8 добавлен пункт «Самостоятельная работа», куда включены задания повышенной сложности;
- в разделе «8. Строки» отдельно рассмотрены приемы работы со строками как с одномерными массивами и использование стандартных процедур и функций для работы со строками;
- в раздел «9. Решение практических задач» включен пункт «Алгоритмы нахождения наибольшего общего делителя двух натуральных чисел»;
- в разделе 10 сделан акцент на основные приемы работы в интегрированной среде и на отладку программы на этапе разработки.

Кроме того, в пособии устранены неточности, имевшие место во втором издании.

Авторы благодарят за помощь, поддержку и критически замечания в работе над пособием декана факультета компьютерных наук и информационных технологий Саратовского государственного университета им. Н.Г. Чернышевского Федорову Антонину Гавриловну и зав. лабораторией системного программирования СГУ Семенову Татьяну Владимировну.

1. БАЗОВЫЕ ЭЛЕМЕНТЫ И СТРУКТУРА ЯЗЫКА TURBO PASCAL

1.1. Алфавит языка Turbo Pascal

Алфавит – совокупность допустимых в языке символов. Алфавит языка Turbo Pascal (TP) можно условно разбить на следующие группы: символы, используемые в идентификаторах; разделители; специальные символы; зарезервированные слова.

Символы, используемые в идентификаторах

Идентификаторы в TP – это имена констант, переменных, меток, типов, объектов, процедур, функций, модулей, программ и полей в записях. Идентификаторы могут иметь произвольную длину, но значащими являются только первые 63 символа.

Идентификатор может включать буквы, цифры и символ подчеркивания. Буквы и символ подчеркивания могут стоять в любой позиции, цифры – в любой, кроме первой.

В качестве букв можно использовать 26 латинских букв, прописных и строчных. Прописные и строчные буквы в идентификаторах не различаются. В качестве цифр используются арабские цифры от 0 до 9.

Замечание. Помимо идентификаторов латинские буквы могут использоваться в записи шестнадцатеричных чисел для обозначения цифр от 10 до 15 (буквы A, B, C, D, E, F или a, b, c, d, e, f), строковых константах, служебных словах и комментариях. В строках символов и комментариях строчные и прописные буквы различаются. Цифры также используются в изображении числовых констант.

Разделители

Разделители используются для отделения друг от друга идентификаторов, чисел, зарезервированных слов. В качестве разделителей можно использовать: пробел, любой управляющий символ (коды от 0 до 31), комментарий. Комментарии заключаются либо в скобки { }, либо в скобки (* *) и могут занимать любое число строк.

Специальные символы

Специальные символы – это знаки пунктуации: + - * / = , ' . : ; < > [ ] ( ) { } ^ @ $ # и знаки операций: <> <= >= :=

Зарезервированные слова

Зарезервированные слова – это служебные слова (например, begin, end, array, function и т.д.) и имена директив. Служебные слова можно использовать только по своему прямому назначению и нельзя использовать в качестве идентификаторов.

Замечание. Русские буквы не входят в алфавит языка, но их можно использовать в комментариях и при формировании символьных констант.

1.2. Структура программы

Программа, написанная на языке TP, имеет следующие разделы:

1) Заголовок программы, который имеет следующий вид: *program* <имя программы>.
2) Раздел описания, который может включать в себя описание констант, переменных, типов, меток, процедур, функций, которые используются в программе.

3) Тело программы или раздел операторов, который начинается со служебного слова *begin* и заканчивается служебным словом *end*. В этом разделе задаются действия над объектами программы, которые вводятся в разделе описаний. Операторы в этом разделе отделяются друг от друга точкой с запятой. После последнего слова *end* ставится точка.

Пример. Программа, вычисляющая сумму двух чисел.

```
program example1_1;      {заголовок программы}
 var x,y,s:integer;      {раздел описания переменных}
begin                    {начало раздела операторов}
 writeln('Введите два числа'); {вывод сообщения на экран}
 readln(x,y);            {ввод двух чисел}
 s:=x+y;                 {подсчет суммы двух чисел}
 writeln('s=',s);        {вывод результата}
end.                     {окончание раздела операторов}
```

Имя этой программы *example1_1*. Из разделов описания присутствует только раздел описания переменных. Он начинается со служебного слова *var*, после которого идет последовательность объявлений переменных, разделенных точкой с запятой. В каждом объявлении перечисляются через запятую имена переменных одного типа, после чего ставится двоеточие и указывается тип переменных. В нашем примере описаны три переменные: *x, y, s*. Все они имеют целый тип *(integer)*, то есть переменные этого типа – это целые числа.

После описательной части идет раздел операторов. Он начинается со служебного слова *begin*, после которого идут операторы языка. Первый оператор - *writeln* – оператор вывода. После запуска программы на выполнение на экране появится сообщение «Введите два числа». *ln* добавляется в конце этого оператора для того, чтобы курсор автоматически переходил на следующую строку после вывода.

Следующий оператор *readln* – оператор ввода. После запуска программы на выполнение необходимо будет ввести два целых числа через пробел (или через Enter - "ввод"), тогда переменной *x* присваивается значение, равное первому введенному числу, а переменной *y* присваивается значение, равное второму введенному числу.

После этих двух операторов стоит оператор присваивания (:= - это знак присваивания). При выполнении этого оператора переменная *s* получит значение, равное сумме чисел *x* и *y*. Так как в результате сложения двух целых чисел получается целое число, то переменная *s* описана типом *integer*.

Последний оператор – это опять оператор вывода. Он выведет на экран текст, заключенный между апострофами, а за ним значение переменной *s*.

1.3. Типы данных

В TP можно выделить следующие группы типов: простые, структурированные, указатели, процедурные типы, объекты. Среди типов, используемых в языке, есть стандартные и определяемые программистом. К стандартным типам, не требующим предварительного определения, относятся целые типы, вещественные типы, логические типы, символьный тип, тип-строка, текстовый файл, указатель. Все другие используемые типы должны быть определены либо в разделе объявления типов, либо в разделе объявления переменных или типизированных констант.

Раздел объявления типов начинается зарезервированным словом *type*, после которого определяются вводимые типы. Определение каждого типа начинается с его имени, затем идет знак равенства, а далее само определение. Определения типов разделяются точкой с запятой. Например:

Type ar=array[1..10] of integer;

где *ar* – имя типа, *array[1..10] of integer* – определение типа.

Рассмотрим некоторые простые типы данных. В TP таковыми являются: целые типы, логический тип, символьный тип, перечисляемый тип, ограниченный тип, вещественные типы.

Целые типы. В TP имеется 5 стандартных типов целых чисел (табл. 1.1). Различаются они диапазоном, наличием или отсутствием знака, а также размером занимаемой памяти.

Таблица 1.1.

Тип	Диапазон	Формат	Размер в байтах
Shortint	-128 .. 127	Знаковый	1
Integer	-32768 .. 32767	Знаковый	2
Longint	-2147483648 .. 2147483647	Знаковый	4
Byte	0 .. 255	Беззнаковый	1
Word	0 .. 65535	Беззнаковый	2

Вещественные типы. В TP имеется 5 стандартных типов вещественных чисел. Их характеристики приведены в таблице 1.2.

Таблица 1.2.

Тип	Диапазон	Число значащих цифр	Размер в байтах
Real	2.9×10^{-39} .. 1.7×10^{38}	11-12	6
Single	1.5×10^{-45} .. 3.4×10^{38}	7-8	4
Double	5.0×10^{-324} .. 1.7×10^{308}	15-16	8
Extended	3.4×10^{-4932} .. 1.1×10^{4932}	19-20	10
Comp	-9.2×10^{18} .. 9.2×10^{18}	19-20	8

Замечание. Для типов Real, Single, Double, Extended диапазон допустимых значений указан по модулю.

Значение действительного типа может быть представлено в двух видах: числом с фиксированной и плавающей точкой. Число с фиксированной точкой изображается десятичным числом с дробной частью (дробная часть может быть нулевой). Дробная часть отделяется от целой с помощью точки. Например, 12.7, 25.00, -185.987, 0.55.

Число с плавающей точкой имеет вид *mEp*, где *m* – мантисса, *p* – порядок числа. В качестве *m* могут быть целые числа и действительные числа с фиксированной точкой, в качестве *p* – только целые числа. Как мантисса, так и порядок, могут содержать знаки +, – (табл. 1.3).

Таблица 1.3.

Математическая запись	Запись с плавающей точкой
0,000007	7E-6
0.62×10^{4}	0.62E+4
$-10,8\times10^{12}$	-10.8E12
20×10^{-3}	20E-3

Таблица 1.4.

Математическая запись	Вывод без указания формата
4	4.0000000000E+00
0.5	5.0000000000E-01
0.0078	7.8000000000E-03
345.2	3.4520000000E+02

Вывод данных вещественного типа допускается с форматом и без него (табл. 1.4). Если при выводе данных вещественного типа отсутствует формат, то число выводится с плавающей точкой – мантисса и порядок. На изображение числа отводится 17 позиций, при этом в целой части мантиссы присутствует только одна значащая цифра.

Изменить стандартную форму вывода можно, используя формат: *write(x:m:n)*, где *x* – выводимое данное вещественного типа, *m* – общее поле выводимого числа (включая знак числа, целую часть, точку и дробную часть), *n* – поле дробной части. В качестве *m* и *n* могут быть целые константы, переменные, выражения. При использовании форматов число выводится в форме с фиксированной точкой. Так, использование формата *write(x:10:4)* для вывода значения *x*, равного 68.486591245, приводит к выводу значения 68.4866.

Логический тип. Стандартный логический тип *boolean* представляет собой тип данных, любой элемент которого может принимать два значения *true* (истина) и *false* (ложь). Занимает в памяти 1 байт.

Символьный тип. Символьный тип данных *char* – это тип данных, элементами которого являются символы: буквы, цифры, знаки препинания и специальные символы. Каждому символу алфавита соответствует индивидуальный числовой код от 0 до 255. Занимает в памяти 1 байт.

Замечание. Наиболее распространенной международной согласованной системой кодирования всех символов является система ASCII. Символы с кодами от 0 до 127 представляют так называемую основную таблицу кодов ASCII. Эта часть идентична на всех IBM-совместимых компьютерах. Коды с символами от 128 до 255 представляют собой национальную часть.

Перечисляемый тип. Перечисляемый тип не является стандартным и задается в виде перечисления некоторых значений. Эти значения образуют упорядоченное множество. Например:

```
type operation=(plus,minus,mult,divide);
```

В этом случае *plus<minus<mult<divide*.

Ограниченный тип. Ограниченный тип данных представляет собой интервал значений порядкового типа. Описание ограниченного типа задают наибольшее и наименьшее значения, входящие в этот интервал. Например,

```
type  chisla=1..25;
      ch='a'..'z';
```

Замечание. Целый, логический, символьный, ограниченный и перечисляемый типы относятся к порядковым типам. Под порядковым типом понимают тип данных, областью значений которых является упорядоченное счетное множество (Счетное множество – это множество, элементы которого можно перенумеровать. Так, например, множество натуральных чисел – счетное, множество действительных чисел – несчетное). Каждому элементу этого множества соответствует число, являющееся его номером при перечислении.

1.4. Константы

Константами называются параметры программы, значения которых задаются только один раз на этапе описания и не меняются в процессе ее выполнения. В TP имеется две разновидности констант:

- обычные константы, тип которых определяется их значением;
- типизированные константы, для которых в явном виде указывается их тип.

Обычные константы могут быть целого, вещественного, символьного, логического и строкового типа. При описании таких констант задаются их имена и значения, разделенные знаком равенства.

Пример

```
const  maxint = 32767;     {целочисленная константа}
       x = -0.5; y=-1e-5;  {вещественные константы}
       symbol = 's';       {символьная константа}
       symbols = 'turbo';  {строковая константа}
```

Типизированные константы могут быть любого типа, кроме типа файл (или типа, содержащего компоненту типа файл). Для каждой такой константы задается ее имя, тип и начальное значение. Тип от имени отделяется двоеточием, начальное значение от типа – знаком равенства.

Пример

```
const  minInt: integer = -32768;
       z: real = -5.089;
       s: char = 's';
       stroka: string  = 'pascal';
```

1.5. Стандартные функции

Арифметические функции. Арифметические функции (табл. 1.5) можно использовать только с величинами целого и вещественного типа.

Таблица 1.5

Функция	Назначение	Тип результата
abs(x)	абсолютное значение аргумента	совпадает с типом аргумента
sqr(x)	квадрат аргумента	совпадает с типом аргумента
sqrt(x)	квадратный корень аргумента	вещественный
cos(x)	косинус аргумента	вещественный
sin(x)	синус аргумента	вещественный
arctan(x)	арктангенс аргумента	вещественный
exp(x)	e^x	вещественный
ln(x)	натуральный логарифм	вещественный
int(x)	целая часть числа	вещественный
frac(x)	дробная часть числа	вещественный

Функции преобразования типов. Эти функции предназначены для преобразования типов величин, например, символа в целое число, вещественного числа в целое и т.д. Такими функциями являются:

Ord(x) – возвращает порядковый номер аргумента и, таким образом, преобразует величину порядкового типа в величину целого типа;

Round(x) – округляет вещественное число до ближайшего целого;

Trunc(x) – выдает целую часть вещественного числа, отбрасывая дробную.

Функции для величин порядкового типа

Odd(x) – проверяет аргумент на нечетность. Аргумент функции величина типа *longint*, результат *true*, если аргумент нечетный, *false* – если четный.

Pred(x) – определяет предыдущее значение величины *x*.

Succ(x) – определяет последующее значение величины *x*.

Ord(x) – возвращает порядковый номер величины *x*.

Пример

ord('a')=97; round(5.47)=5; trunc(5.47)=5; odd(51)=true; pred(50)=49;
ord(1)=1; round(5.74)=6; trunc(5.74)=5; odd(24)=false; succ(50)=51

1.6. Знаки операций

Все операции в ТР можно разбить на группы. Мы рассмотрим три из них: арифметические операции, логические операции и операции отношения.

Арифметические операции. Арифметические операции (табл. 1.6) применимы только к величинам целых и вещественных типов.

Таблица 1.6

Знак	Операция	Типы операндов	Тип результата
+	Сложение	целые хотя бы один вещественный	целый вещественный
-	Вычитание	целые хотя бы один вещественный	целый вещественный
*	Умножение	целые хотя бы один вещественный	целый вещественный
/	Деление	целые или вещественные	вещественный
div	целая часть от деления целых чисел	целые	целый
mod	остаток от деления целых чисел	целые	целый

Знаки операций +, - , * используются также и с некоторыми другими типами операндов, но тогда они имеют другой смысл. В операциях деления делитель не должен равняться 0. При использовании знака операции, являющегося служебным словом, он должен быть отделен от операндов хотя бы одним пробелом.

Пример

17 div 4 = 4; 17 mod 4 = 1; -21 div 4 = -5; -21 mod 4 = -1.

Логические операции. Логические операции (табл. 1.7) применяются к величинам логического типа, результат операции – тоже логического типа. Имеется одна унарная логическая операция *not* (отрицание) и три бинарные операции *and* (и), *or* (или), *xor* (исключающее или).

Таблица 1.7

Значение операнда		Значение операции			
x	y	not x	x and y	x or y	x xor y
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

Пример

Пусть *a=5, b=-3, c=true.*

Тогда значением выражения *(a>0) and (a<10) and (b>-10)* будет *true*, а значением выражения *(b>0) or not c* будет *false*.

Операции отношения. Операции отношения предназначены для сравнения двух величин. Результат сравнения имеет логический тип.

= - равно < - меньше <= - меньше или равно
<> - не равно > - больше >= - больше или равно.

Порядок вычисления выражений. Выражение – это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражение формируется из констант, переменных, функций, знаков операций и круглых скобок. Вычисление значений выражений выполняется в определенном порядке. В первую очередь вычисляются выражения, заключенные в круглые скобки (таким образом, круглые скобки используются для заключения в них той части выражения, которую нужно выполнить в первую очередь). Для любых вложенных друг в друга пар круглых скобок вычисляется сначала внутреннее выражение, а затем внешнее. Далее вычисляются значения входящих в выражение функций и т.д. Приоритеты выполняемых действий таковы:

1. Вычисления в круглых скобках.
2. Вычисления значений функций.
3. Унарные операции –, *not*.
4. Операции *, /, *div, mod, and*.
5. Операции +, -, *or, xor*.
6. Операции отношения >, <, >=, <= <>, =.

Примеры

1. Выражение $\frac{x^2+\sin(x+1)}{25}$ запишется по правилам ТР следующим образом: (x*x+sin(x+1))/25.

2. (x > 1) and (x < 5) or (x > 10) and (x <20)

Сначала вычисляются слева направо четыре операции сравнения, помещенные в круглые скобки, затем две логические операции and и в последнюю очередь выполняется операция OR.

3. (x > 1) and ((x < 5) or (x > 10)) and (x <20)

Сначала вычисляются четыре операции сравнения, помещенные в круглые скобки, затем логическая операция or и в последнюю очередь выполняются операции and.

1.7. Совместимость и преобразование типов данных

Совместимость типов учитывается при вычислении выражений и выполнении операторов присваивания. Несовместимость типов определяется на этапе компиляции программы; при этом выдается сообщение об ошибке. Если типы операндов выражения не одинаковы, но совместимы, производится преобразование типов для приведения их к одному допустимому типу. Во время вычисления выражений два типа операндов совместимы если:

1) оба они одного типа;
2) один операнд вещественный, другой – целый;
3) один операнд является диапазоном типа второго операнда;
4) оба операнда – диапазоны одного и того же базового типа;
5) оба операнда – строки;
6) один операнд типа строка, другой – символ.

Если *T1* – тип переменной левой части оператора присваивания; а *T2* – тип результата выражения его правой части, то присваивание возможно если:

1) *T1, T2* – один и тот же тип;
2) *T1, T2* – совместимые порядковые типы и значение *T2* лежит в диапазоне возможных значений *T1*;
3) *T1, T2* – вещественные типы и значение *T2* лежит в диапазоне возможных значений *T1*;
4) *T1* – вещественный тип; *T2* – целый тип;
5) *T1* – строка, *T2* – строка или символ;
6) *T1* и *T2* – совместимые множества и все элементы *T2* принадлежат множеству возможных значений *T1*.

1.8. Примеры простых программ

1. Написать программу, которая находит частное двух целых чисел.

Указания по решению задачи. Следует обратить внимание на то, что результат деления двух целых чисел всегда принадлежит вещественному типу данных.

```
program example1_1;
 var x,y:integer;
     c:real;
begin
 writeln('Введите два числа'); readln(x,y);
 c:=x/y;
 writeln('c=',c:4:1);
end.
```

Результат работы программы:

x	y	c
24	6	4.0
24	5	4.8

2. Составить программу, которая для двух данных целых чисел получает остаток от деления одного на другое.

```
program example1_2;
 var x,y,c:integer;
begin
 writeln('Введите два числа'); readln(x, y);
 c:=x mod y;
 writeln('c=',c);
end.
```

Результат работы программы:

x	y	c
24	6	0
24	5	4

1.9. Упражнения

I. Вычислить:

1) trunc(6.9);	2) round(6.9);	3) trunc(6.2);	4) round(6.2);	5) round(-1.8);
6) round(0.5);	7) trunc(0.5);	8) 20 div 6;	9) 20 div 4;	10) 20 mod 4;
11) 2 div 5;	12) 2 mod 5;	13) 3 mod 3;	14) int(1.3);	15) frac(1.3);
16) int(3.8);	17) trunc(-1.8);	18) 20 mod 6;	19) 123 div 0;	20) frac(3.8).

В каком случае результат будет вещественного типа, а в каком – целого?

II. Объяснить, что будет напечатано программой

1)
```
program e1;
  var b,c,d: real;
begin
  read(b,c); d:=sqrt(sqr(b)-4*c);
  writeln('x1=', (-b+d)/2, '  x2=', (-b-d)/2)
end.
```

Исходные значения a и b: a) 2.0 и 0; b) 2.0 и 1.0?

2)
```
program e2;
 begin
  write(1); write(2, 3);
  writeln(4); write(5, 6);
  writeln; writeln(7,8)
 end.
```

3)
```
program e3;
  var a, b: integer;
 begin
  read(a,b); writeln(a, b, a)
 end.
```

Исходные значения a и b: 2 и 0?

4)
```
program e4;
  var x: integer;
 begin
  read(x); writeln('x+1')
 end.
```

Исходное значение x: a) 12; b) -89?

5)
```
program e5;
  var  fi: real;
       h, m:integer
begin
 read(fi); h:=trunc(fi/30);
 m:=trunc(fi-30*h)/0.5; writeln(h,' ', m)
end.
```

Исходное значение fi: 31.7

6)
```
program e6;
  var x: real; t: boolean;
 begin
  read(x); t:=x<round(x); read(x);
  t:=t and (x<trunc(x)); writeln(t)
 end.
```

В качестве исходных данных заданы числа: a) 3.9 и 1; b) 3.4 и 1?

III. Найти и объяснить ошибки в каждой из следующих программ:

1)
```
program e1;
  const d=5;
begin
  d:=sqr(d);
  writeln('d**2=',d)
end.
```

2)
```
program e2;
  const k=true;
  var x: real;
begin
  read(x); writeln(ord(x)=k);
end.
```

3)
```
program e3;
  var a, b, c: integer;
begin
 read(a,b);  writeln((a+b+c)/3)
end.
```

4)
```
program e4;
  var x:real;
 begin
  read(x); y:=sqrt(x)+1; writeln(y)
 end.
```

5)
```
program e5;
  const b=2.5;
  var a, b, c: real;
begin
 read(a, c); writeln(a*c>b)
end.
```

6)
```
program e6;
  const p=3.14159;
  var a, b: integer;
 begin
  read(a); d:=odd(p*0) and b>c; writeln(d)
 end.
```

IV. Записать по правилам ТР следующие выражения:

1) $10\sin x+|x^4-x^5|$;

2) $e^{-x}-\cos x+\sin 2xy$;

3) $\sqrt{x^4+\sqrt{|x+1|}}$;

4) $\dfrac{\sin x+\cos y}{\operatorname{tg}x}+0{,}43$;

5) $\dfrac{0{,}125x+|\sin x|}{1{,}5x^2+\cos x}$;

6) $\dfrac{x+y}{x+1}-\dfrac{xy-12}{34+x}$;

7) $\dfrac{\sin x+\cos y}{\cos x-\sin y}\operatorname{tg}xy$;

8) $\dfrac{1+e^{y-1}}{1+x^2\,|y-\operatorname{tg}x|}$;

9) $|x^3-x^2|-\dfrac{7x}{x^3-15x}$;

10) $1+\frac{x}{3}+|x|+\frac{x^3+4}{2}$;

11) $\frac{\ln|\cos x|}{\ln(1+x^2)}$;

12) $\frac{1+\sin\sqrt{x+1}}{\cos(12y-4)}$;

13) $\frac{a^2+b^2}{1-\frac{a^3-b}{3}}$;

14) $\frac{1+\sin^2(x+y)}{2+\left|x-\frac{2x}{1+x^2y^2}\right|}+x$;

15) $x\cdot\ln x+\frac{y}{\cos x-\frac{x}{3}}$;

16) $\sin\sqrt{x+1}-\sin\sqrt{x-1}$;

17) $\frac{\cos x}{\pi-2x}+16x\cdot\cos xy$;

18) $2\mathrm{ctg}3x-\frac{1}{12x^2+7x-5}$;

19) $\frac{b+\sqrt{b^2+4ac}}{2a}-a^3+b^{-2}$;

20) $\ln\left|(y-\sqrt{|x|})(x-\frac{y}{x+\frac{x^2}{4}})\right|$.

V. Написать программу, которая подсчитывает:

1) периметр квадрата, площадь которого равна a;
2) площадь равностороннего треугольника, периметр которого равен p;
3) расстояние между точками с координатами a, b и c,d;
4) среднее арифметическое кубов двух данных чисел;
5) среднее геометрическое модулей двух данных чисел;
6) гипотенузу прямоугольного треугольника по двум данным катетам a, b;
7) площадь прямоугольного треугольника по двум катетам a, b;
8) периметр прямоугольного треугольника по двум катетам a, b;
9) площадь полной поверхности куба с ребром a;
10) объем куба с ребром a;
11) периметр треугольника, заданного координатами вершин (x1, y1), (x2, y2), (x3, y3);
12) площадь треугольника, заданного координатами вершин (x1, y1), (x2, y2), (x3, y3);
13) длину окружности и площадь круга с радиусом r;
14) площадь грани и площадь полной поверхности куба с ребром a;
15) объем куба с ребром a;
16) площадь равнобедренной трапеции с основаниями a и b и углом α при большем основании;
17) площадь кольца с внутренним радиусом r1 и внешним r2;
18) радиус окружности, вписанной в равносторонний треугольник со стороной a;
19) радиус окружности, описанной около равностороннего треугольника со стороной a;
20) сумму членов арифметической прогрессии, если известен ее первый член, разность и число членов прогрессии.

1.10. Самостоятельная работа

Задача 1. На уроке информатики учитель задал Пете задачу следующего содержания. На координатной прямой *Ox* заданы координаты трех городов A, B, C. Город A находится в точке с координатой *a*, город B - в точке с координатой *b*, а C

– в точке с координатой *c*. Путешественник проделал путь из города A в город B, а затем из города B в город C. Помогите определить Пете длину пути, проделанного путешественником.

Задача 2. Из трехзначного числа *x* вычли его последнюю цифру. Когда результат разделили на 10, а к частному слева приписали последнюю цифру числа *x*, то получили число *y*. Найдите число *x* по заданному значению *y*. Значение *y* вводится с клавиатуры; 99<y<1000; число десятков у *y* не равно 0.

2. ОПЕРАТОРЫ ВЕТВЛЕНИЯ

2.1. Условный оператор IF

Оператор *if* относится к операторам разветвления процесса обработки данных. Условный оператор *if* может иметь одну из форм: сокращенную (рис. 2.1.а) или полную (рис.2.1.б).

Форма сокращенного оператора: if B then S;

где *B* –логическое выражение; *S* – выполняемый оператор, простой или составной.

а) Схема сокращенной формы б) Схема полной формы

Рис. 2.1 Условный оператор *if*

При выполнении оператора *if* сокращенной формы сначала вычисляется выражение *B*, затем проводится анализ его результата: если *B* истинно (*TRUE*), то выполняется оператор *S*; если *B* ложно (*FALSE*), то оператор *S* пропускается. Таким образом, с помощью сокращенной формы оператора *if* можно выполнить или нет оператор, стоящий после ключевого слова *then*.

Форма полного оператора: if B then S1 else S2;

где *B* –логическое выражение, которое является условием разветвления (принятия решения); *S1*, *S2*- выполняемые операторы, простые или составные.

При выполнении оператора *if* полной формы сначала вычисляется выражение *B*, затем анализируется его результат: если *B* истинно, то выполняется оператор *S1* (ветвь *then*), а оператор *S2* пропускается; если *B* ложно, то выполняется оператор *S2* (ветвь *else*), а *S1* – пропускается. Таким образом, с помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса вычисления.

Замечание. В программе символ «точка с запятой» ставится только в конце оператора. Так как служебные слова *then* и *else* входят в состав оператора *if*, то перед ними точка с запятой не ставится. Кроме того, условие разветвления может включать в себя несколько простых условий (каждое из которых заключается в скобки), соединенных логическими операциями (см. пример 4).

Примеры:

1. {Сокращенная форма}
```
if a > 0 then x:=y;
```
2. {Полная форма с простыми операторами}
```
if a > 0 then x:=y else x:=z;
```

3. {Полная форма с простым оператором в ветви *then* и составным оператором в ветви *else*}

```
if a < 0 then x:=1
        else begin x:=1; y:=2 end;
```

4. {Полная форма с составными операторами}

```
if (a > 0) and (c >0) then begin  x:= 0; y:= 1 end
                      else begin  x:=1; y:=0 end
```

Операторы *S1* и *S2* (см. 2.1.b) могут также являются операторами *if*. Такие операторы называют вложенными. При этом ключевое слово *else* связывается с ближайшим предыдущим словом *then*, которое еще не связано ни с одним *else*. Примеры схем алгоритмов с вложенными условными операторами и соответствующими им текстами программ приведены на рис. 2.2, 2.3.

Пример для схемы рис. 2.2 Уровни вложенности операторов IF:

```
if A< B then
        if C < D then
                if E < F then X := Q
                         else X := R
                else X := Z
else X := Y;
```

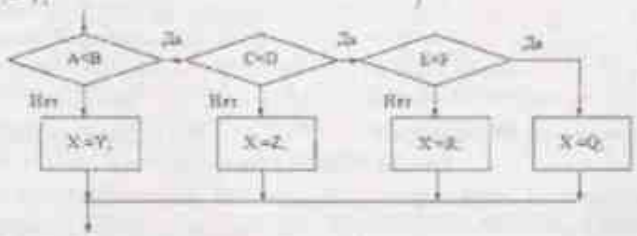


Рис. 2.2. Схема алгоритма с вложенными условными операторами

Пример для схемы рис. 2.3 Уровни вложенности оператора IF:

```
if A < B then
        if C < D then X := Y
                 else X := Z
else
        if E < F then X := R
                 else X := Q;
```

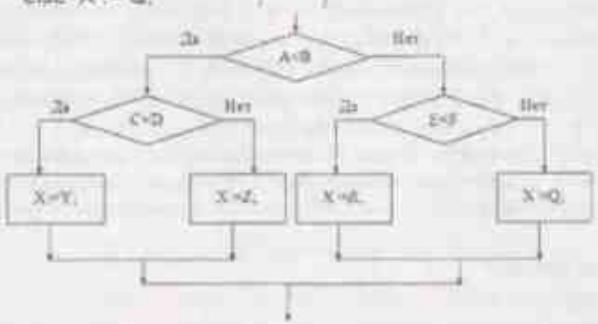


Рис. 2.3. Схема алгоритма с вложенными условными операторами

2.2. Оператор выбора CASE

Данный оператор является обобщением условного оператора *if* для случая произвольного числа альтернатив. Оператор *case*, как и условный оператор *if*, может иметь одну из форм: сокращенную или полную. Схема полной формы приведена на рис. 2.4. Структура оператора:

```
case S of
   C1: instruction1;
   C2: instruction2;
   ...
   CN: instructionN;
 else instruction
end;
```

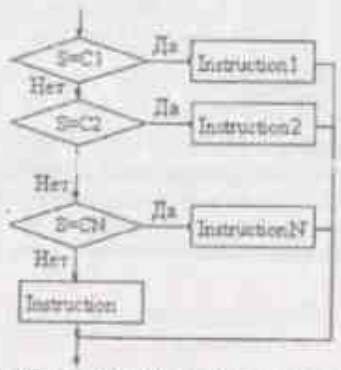


Рис. 2.4.Схема полной формы оператора *Case*.

В этой структуре: *S* – выражение порядкового типа, значение которого вычисляется; *C1, C2, ..., CN* – константы порядкового типа, с которыми сравнивается значение выражения *S*; *Instruction1, Instruction2,..., Instruction* – операторы (простые или составные), из которых выполняется тот, с константой которого совпадает значение выражения *S*; при этом *Instruction* – оператор, который выполняется, если значение выражения S не совпадает ни с одной из констант *C1, C2, ..., CN*.

Ветвь *else* является необязательной. Если она отсутствует (сокращенная форма оператора *case*) и значение выражения *S* не совпадает ни с одной из перечисленных констант, то весь оператор *case* рассматривается как пустой.

Если для нескольких констант нужно выполнить один и тот же оператор, их можно перечислить через запятую или даже указать диапазон (если возможно), сопроводив их общим оператором, например:

```
Readln(i);
Case i of
  0, 2, 4, 6, 8: writeln('Четная цифра');
  1, 3, 5, 7, 9: writeln('Нечетная цифра');
  10..100: writeln('Число от 10 до 100');
 else writeln('Отрицательное число или число больше 100');
end;
```

В данном примере в зависимости от введенного значения *i* выводится сообщение о характере значения этой переменной. Так, если *i=4*, то значение переменной попадает в список констант первой ветки оператора *case*, и на экран выводится сообщение 'Четная цифра'. Если *i=56*, то значение переменной попадает в диапазон констант третьей ветки оператора *case*, и на экран выводится сообщение 'Число от 10 до 100'. Если *i=-56*, то значение переменной не совпадает ни с одной из констант первых трех ветвей оператора, поэтому выполняется оператор ветки *else* и на экран выводится сообщение 'Отрицательное число или число больше 100'.

2.3. Примеры использования операторов ветвления при решении задач

1. Выбрать наименьшее число из двух вещественных чисел x, y.

Указания по решению задачи. Для того, чтобы число x было наименьшим (минимальным) необходимо, чтобы выполнялось условие x<y. Если это условие ложно, то либо y является наименьшим, либо x и y равны (в этом случае в качестве наименьшего значения можно вывести любое их заданных чисел).

```
program example2_1;
  var x, y, min: real;
begin
 writeln('Введите значение x и y'); readln(x, y);
 if x<y        {проверка условия минимальности x }
  then min:=x {если условие истинно, то минимальное значение = x}
  else min:= y {если условие ложно, то минимальное значение =y}
 writeln('Минимальное значение = ', min:6:2);
end.
```

Результат работы программы:

x	*y*	*min*
0.3	-0.3	-0.3
0.4	1.50	0.4

2. Дано трехзначное натуральное число *a*. Определить наибольшую цифру числа.

Указания по решению задачи. Сначала необходимо разложить число a на составляющие цифры. Это можно сделать с помощью арифметических операций *div* и *mod*. Затем из трех цифр нужно выбрать наибольшую. Для того чтобы *x* было наибольшей (максимальной) из цифр *x, y, z*, необходимо чтобы выполнялось условие $(x \geq y)$ *и* $(x \geq z)$. Если это условие ложно, то наибольшей цифрой может быть либо y, либо z и их нужно сравнить между собой. В этом случае, если выполниться условие $y \geq z$, то наибольшей цифрой будет *y*, иначе *z*. С учетом сказанного программа выглядит следующим образом:

```
program example2_2;
  var a: word;
      x, y, z, max: byte;
begin
 writeln('Введите значение a'); readln(a);
 z:=a mod 10;        {находим первую справа цифру числа a}
 a:=a div 10;        {отбрасываем от числа первую цифру слева}
 y:= a mod 10;       {находим среднюю цифру исходного числа a}
 x:= a div 10;       {находим первую слева цифру исходного числа a}
 if (x >= y) и (x >= z)      {проверка условия максимальности x}
  then max:=x        {если условие истинно, то максимально x}
  else if y >= z     {сравниваем между собой y и z и находим наибольшую из них}
       then max:=y  else max:=z;
 writeln('Наибольшая цифра в числе = ', max);
end.
```

Результат работы программы:

a	*наибольшая цифра*
123	3
132	3
312	3

3. Для произвольных значений аргументов вычислить значение функции, заданной следующим образом: $y(x)=\frac{1}{x}+\sqrt{x+1}$. Если в некоторой точке вычислить значение функции окажется невозможно, то вывести на экран сообщение «функция не определена».

Указания по решению задачи. Данную задачу можно решать двумя способами.

I способ. Заданная функция не определена в том случае, когда знаменатель первого слагаемого равен нулю или подкоренное выражение второго слагаемого имеет отрицательное значение, т.е x=0 или x+1<0, что в Паскале эквивалентно условию *(x=0) or (x<-1)*. В остальных случаях функция определена и, следовательно, ее значение в заданной точке может быть подсчитано и выведено на экран. С учетом сказанного программа выглядит следующим образом:

```
program example2_3;
  var x, y: real;
begin
 writeln('Введите значение x'); readln(x);
 if (x = 0) or (x < -1)        {проверка условия неопределенности функции }
  then writeln('Функция не определена')
  else begin  y := 1/x +sqrt(x+1);  writeln('y(',x,') = ', y:6:2);  end;
end.
```

II способ. Заданная функция определена в том случае, когда знаменатель первого слагаемого не равен нулю и подкоренное выражение второго слагаемого неотрицательно, т.е. x≠0 и x+1≥0, что в ТР эквивалентно условию *(x<>0) and (x>=-1)*. В остальных случаях функция не определена. С учетом сказанного программа выглядит следующим образом:

```
program example2_4;
  var x, y: real;
begin
 writeln('Введите значение x'); readln(x);
 if (x <> 0) and (x >= -1)        {проверка условия определенности функции }
  then begin  y := 1/x +sqrt(x+1);  writeln('y(',x,') = ', y:6:2);  end
  else writeln('Функция не определена');
end.
```

Обе программы дадут нам следующий результат:

x	*y(x)*
0	функция не определена
2	1.50

3. Для произвольных значений аргументов вычислить значение функции, заданной следующим образом: $y(x)=\begin{cases}\left(x^2+1\right)^2, & \text{при } x<0;\\ 0, & \text{при } 0\le x<1;\\ \left|x^2-5x+1\right|, & \text{при } x\ge 1.\end{cases}$

Указания по решению задачи. Вся числовая прямая Ox разбивается на три непересекающихся интервала, (-∞; 0), [0; 1), [1; +∞). На каждом интервале функция задается своею ветвью. Заданная точка x может попасть только в один из указанных интервалов. Чтобы определить, в какой из интервалов попала точка, пользуемся следующим алгоритмом. Если *x<0*, то *x* попадает в первый интервал, и функцию вы-

считываем по первой ветви, после чего проверка заканчивается. Если это условие ложно, то истинно условие $x \geq 0$, и для того чтобы точка попала во второй интервал достаточно, чтобы выполнялось условие $x < 1$. Если выполняется это условие, то точка x попадает во второй интервал и мы определяем функцию по второй ветви, после чего заканчиваем вычисления. В противном случае, точка может принадлежать только третьему интервалу, поэтому дополнительная проверка не проводится, а сразу вычисляем функцию по третьей ветви. Приведенный алгоритм можно реализовать с помощью вложенных операторов *if*.

```
program exampel2_5;
  var x, y: real;
begin
 writeln('Введите значение x'); readln(x);
 if x<0          {Проверяем условие первой ветви}
  then y := sqr(sqr(x)*x + 1)
  else if x<1          {Проверяем условие второй ветви}
       then y := 0  else y := abs(sqr(x) – 5*x +1);
 writeln('y(',x,') = ', y:6:2);
end.
```

Результат работы программы:

координата точки	*ответ*
0	0
1	3
-2	49

4. Дана точка на плоскости с координатами (x, y). Составить программу, которая выдает одно из сообщений «Да», «Нет», «На границе» в зависимости от того, лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе.

Указания по решению задачи. Заданная область разбивает всю плоскость на три непересекающихся множества точек. В общем случае эти множества обозначим следующим образом: I_1 – множество точек, лежащих внутри области; I_2 – множество точек, лежащих вне области; I_3 – множество точек, образующих границу области. Точка с координатами (x, y) может принадлежать только одному из них. Поэтому проверку можно проводить по аналогии с алгоритмом, приведенном в примере 3. Однако множества I_1, I_2, I_3 значительно труднее описать математически, чем интервалы в примере 3. Поэтому для непосредственной проверки выбираются те два множества, которые наиболее просто описать математически. Обычно труднее всего описать точки границы области. Например, для рис. 2.5 множества задаются следующим образом:

I_1: $x^2 + y^2 < 10^2$; I_2: $x^2 + y^2 > 10^2$; I_3: $x^2 + y^2 = 10^2$.

Для рис. 2.6 множества задаются следующим образом:

I_1: $|x| < 10$ и $|y| < 5$; I_2: $|x| > 10$ или $|y| > 5$;

I_3: ($|x| <= 10$ и $y = 5$) или ($|x| <= 10$ и $y = -5$) или ($|y| <= 5$ и $x = 10$) или ($|y| <= 5$ и $x = -10$).

рис. 2.5

рис. 2.6

Как видим для рис. 2.5 описание всех множеств равносильно по сложности, а для рис. 2.6 описать множество I_3 значительно сложнее.

```
program exampel2_6; {для рис.2.5}
var x,y:real;
begin
 writeln('Введите координаты точки'); readln(x,y);
 if sqr(x)+sqr(y)<100      {точка внутри области?}
  then writeln('Да')
  else if sqr(x)+sqr(y)>100   {точка вне области ?}
       then writeln('Нет')  else writeln('На границе')
end.
```

Результаты работы программы:

координаты точек		*ответ*
0	0	да
10	0	на границе
-12	13	нет

```
program exampel2_7; {для рис.2.6}
var x,y:real;
begin
writeln('Введите координаты точки'); readln(x,y);
if (abs(x)<10) and (abs(y)<5) {точка внутри области?}
 then writeln('Да')
 else if (abs(x)>10) or (abs(y)>5) {точка вне области?}
      then writeln('Нет') else writeln('На границе')
end.
```

Результаты работы программы:

координаты точек		*ответ*
0	0	да
10	5	на границе
-12	13	нет

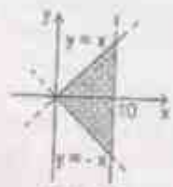


рис. 2.7.

```
program exampel2_8;
var x,y:real;
begin
 writeln('Введите координаты точки'); readln(x,y);
  if (y < x) and ( y > -x) and (x <10) {точка внутри области?}
     then writeln('Да')
     else if (y > x) or ( y < -x) or (x >10) {точка вне области?}
             then writeln('Нет') else writeln('На границе')
end.
```

Результаты работы программы:

координаты точек		*ответ*
0	0	на границе
9	0	да
-12	13	нет

5. Составить программу, которая в зависимости от порядкового номера дня недели (1, 2, …, 7) выводит на экран его название (понедельник, вторник, …, воскресенье).

```
program example2_9;
  var I: byte;
begin
 writeln('Введите номер дня недели'); readln(I);
 Case I of
    1: writeln('понедельник');
    2: writeln('вторник');
    3: writeln('среда');
    4: writeln('четверг');
    5: writeln('пятница');
    6: writeln('суббота');
    7: writeln('воскресенье');
 else writeln('ОШИБКА');
 end;
end.
```

Результаты работы программы:

Номер дня недели	*ответ*
4	четверг
-2	ОШИБКА

2.4. Упражнения

I. Объяснить, что будет напечатано каждой программой, если в качестве исходных данных будет введено значение: a) 0; b) 5; c) 10

```
1) program e1;
   var x, y: byte;
  begin
   read(x);
   if x<5 then y:=1 else y:=2;
   if x>5 then y:=3 else y:=4;
   writeln('y=', y)
  end.
```

```
2) program e2;
   var x, y: byte;
  begin
   read(x);
    if x<5 then y:=-1 else
    if x>5 then y:=0 else y:=1;
   writeln('y=', y)
  end.
```

```
3) program e3;
   var x, y: byte;
  begin
   read(x);
   y:=x mod 2;
   if not (y=0)
     then write ('yes')
     else write('no');
end.
```

```
4) program e4;
   var x, y: byte;
  begin
   read(x);
   case x of
    0, 2: y:=1;
    1, 3..9: y:=2;
    else y:=3;
   end;
   write('y=', y);
  end.
```

II. Найти и объяснить ошибки в каждом фрагменте программы:

```
1) if odd(x)
    then write('число четное')
    else write('число нечетное');
```

```
2) if odd(x)
    then write('число нечетное');
    else write('число четное');
```

```
3) if x>=0 and x<10
    then write('введена цифра');
```

```
4) if x mod 2
    then write('число нечетное'); else write('число четное');
```

```
5) case x/2 of
    0: write('число четное');
    1: write('число нечетное');
   end;
```

```
6) case x of
    0...9: write('введена цифра');
    10...99: write('введено двухзначное число');
   end;
```

III. Используя операции not, and, or, xor записать сложные условия, истинные для следующих ситуаций:

1) число x принадлежит отрезку [0, 10];
2) число x лежит вне отрезка [-3, 3];
3) число x принадлежит интервалу [-1, 1];
4) хотя бы одно из чисел x, y, z является положительным;
5) только одно из чисел x, y, z является отрицательным;
6) все три числа x, y, z равны;
7) все три числа x, y, z не равны;
8) только два из трех чисел x, y, z равны;
9) точка с координатой (x, y) лежит внутри единичного круга с центром в начале координат;
10) точка с координатой (x, y) лежит вне единичного круга с центром в точке (2, 4);
11) число x делится на y без остатка;
12) хотя бы одно из чисел x, y не делится на z;

13) цифры двухзначного числа x равны;
14) старшая цифра трехзначного числа x на 1 меньше младшей цифры;
15) сумма целых чисел x и y является трехзначным числом, оканчивающимся на 0.

IV. Дано сложное логическое условие. На координатной плоскости изобразите графически множество точек, для которых это условие является истинным.

1) (x>0) and (y>0)
2) (x>0) or (y>0)
3) (x>0) xor (y>0)
4) (y>0) and (y<5) and (x=2)
5) (x<0) and (y<0) and (x+y>0)
6) (abs(x)>2) or (abs(y)>2)
7) (x=1) xor (x=3)
8) (abs(x)<1) and (abs(y)<1) and (x*x+y*y>1)
9) (x*x+y*y>1) and ((x>0.5) or (x<-0.5))
10) ((abs(x)<1) and (abs(y)<1)) or (x*x+y*y>10)

V. Написать программу, которая определяет:

1) максимальное значение для двух различных вещественных чисел;
2) является ли заданное целое число четным;
3) является ли заданное целое число нечетным;
4) если целое число M делится на целое число N, то на экран выводится частное от деления, в противном случае выводится сообщение «M на N нацело не делится»;
5) оканчивается ли данное целое число цифрой 7;
6) имеет ли уравнение $ax^2+bx+c=0$ решение, где a, b, c – данные вещественные числа;
7) какая из цифр двухзначного числа больше: первая или вторая;
8) одинаковы ли цифры данного двухзначного числа;
9) является ли сумма цифр двухзначного числа четной;
10) является ли сумма цифр двухзначного числа нечетной;
11) кратна ли трем сумма цифр двухзначного числа;
12) кратна ли числу A сумма цифр двухзначного числа;
13) какая из цифр трехзначного числа больше: первая или последняя;
14) какая из цифр трехзначного числа больше: первая или вторая;
15) какая из цифр трехзначного числа больше: вторая или последняя;
16) все ли цифры трехзначного числа одинаковые;
17) существует ли треугольник с длинами сторон a, b, c;
18) является ли треугольник с длинами сторон a, b, c прямоугольным;
19) является ли треугольник с длинами сторон a, b, c равнобедренным;
20) является ли треугольник с длинами сторон a, b, c равносторонним.

VI. Для произвольных значений аргументов вычислить значение функции, заданной следующим образом:

1. $y=\frac{1}{(1+x)^2}$;
2. $y=\frac{1}{x^2-1}$;
3. $y=\sqrt{x^2-1}$;
4. $y=\sqrt{5-x^3}$;
5. $y=\ln(x-1)$;
6. $y=\ln(4-x^2)$;
7. $y=\frac{x}{\sqrt{2x-1}}$;
8. $y=\frac{3x+4}{\sqrt{x^2+2x+1}}$;
9. $y=\frac{1}{x-1}+\frac{2}{1-4x}$;

10. $y = \ln|x-2|$;

11. $y = \ln\dfrac{x}{x-2}$;

12. $y = \ln(x^4-1)\ln(1+x)$;

13. $y = \dfrac{\ln(x-2)}{\sqrt{5x+1}}$;

14. $y = \dfrac{\sqrt{x^2-2x+1}}{\ln(4-2x)}$;

15. $y = \ln|3x\sqrt{2x^3-1}|$;

16. $y = \dfrac{3}{|x^3+8|}$;

17. $y = \dfrac{x+4}{x^2-2} + \sqrt{x^3-1}$;

18. $y = \sqrt{x^2+1} - \sqrt{x^2+5}$;

19. $y = \dfrac{\sqrt{x^3-1}}{\sqrt{x^2-1}}$;

20. $y = \dfrac{1}{x+7} + \ln(1-|x|)$.

Замечание. Если в некоторой точке вычислить значение функции окажется невозможно, то вывести на экран сообщение «функция не определена».

VII. Для произвольных значений аргументов вычислить значение функции, заданной следующим образом:

1. $y = \begin{cases} \dfrac{1}{(0.1+x)^2}, & \text{если } x \ge 0.9; \\ 0.2x+0.1, & \text{если } 0 \le x < 0.9; \\ x^2+0.2, & \text{если } x < 0. \end{cases}$

2. $y = \begin{cases} \sin(x), & \text{если } |x| < 3; \\ \dfrac{\sqrt{x^2+1}}{\sqrt{x^2+5}}, & \text{если } 3 \le |x| < 9; \\ \sqrt{x^2+1} - \sqrt{x^2+5}, & \text{если } |x| \ge 9. \end{cases}$

3. $y = \begin{cases} 0, & \text{если } x < a; \\ \dfrac{x-a}{x+a}, & \text{если } x > a; \\ 1, & \text{если } x = a. \end{cases}$

4. $y = \begin{cases} x^3-0.1, & \text{если } |x| \le 0.1; \\ 0.2x-0.1, & \text{если } 0.1 < |x| \le 0.2; \\ x^3+0.1, & \text{если } |x| > 0.2. \end{cases}$

5. $y = \begin{cases} a+b, & \text{если } x^2-5x < 0; \\ a-b, & \text{если } 0 \le (x^2-5x) < 10; \\ ab, & \text{если } x^2-5x \ge 10. \end{cases}$

6. $y = \begin{cases} x^2, & \text{если } (x^2+2x+1) < 2; \\ \dfrac{1}{x^2-1}, & \text{если } 2 \le (x^2+2x+1) < 3; \\ 0, & \text{если } (x^2+2x+1) \ge 3. \end{cases}$

7. $y = \begin{cases} -4, & \text{если } x < 0; \\ x^2+3x+4, & \text{если } 0 \le x < 1; \\ 2, & \text{если } x \ge 1. \end{cases}$

8. $y = \begin{cases} x^2-1, & \text{если } |x| \le 1; \\ 2x-1, & \text{если } 1 < |x| \le 2; \\ x^5-1, & \text{если } |x| > 2. \end{cases}$

9. $y = \begin{cases} (x^2-1)^2, & \text{если } x < 1; \\ \dfrac{1}{(1+x)^2}, & \text{если } x > 1; \\ 0, & \text{если } x = 1. \end{cases}$

10. $y = \begin{cases} x^2, & \text{если } (x+2) \le 1; \\ \dfrac{1}{x+2}, & \text{если } 1 < (x+2) < 10; \\ x+2, & \text{если } (x+2) \ge 10; \end{cases}$

11. $y = \begin{cases} x^2+5, & \text{если } x \le 5; \\ 0, & \text{если } 5 < x < 20; \\ 1, & \text{если } x \ge 20. \end{cases}$

12. $y = \begin{cases} 0, & \text{если } x < 0; \\ x^2+1, & \text{если } x \ge 0 \text{ и } x \ne 1; \\ 1, & \text{если } x = 1. \end{cases}$

13. $y = \begin{cases} 1, & \text{если } x = 1 \text{ или } x = -1; \\ \dfrac{-1}{1-x}, & \text{если } x \ge 0 \text{ и } x \ne 1; \\ \dfrac{1}{1+x}, & \text{если } x < 0 \text{ и } x \ne -1. \end{cases}$

14. $y = \begin{cases} 0.2x^2-x-0.1, & \text{если } x < 0; \\ \dfrac{x^2}{x-0.1}, & \text{если } x > 0 \text{ и } x \ne 0.1; \\ 0, & \text{если } x = 0.1. \end{cases}$

15. $y = \begin{cases} 1, & \text{если } (x-1) < 1; \\ 0, & \text{если } (x-1) = 1; \\ -1, & \text{если } (x-1) > 1. \end{cases}$

16. $y = \begin{cases} x, & \text{если } x > 0; \\ 0, & \text{если } -1 \le x \le 0; \\ x^2, & \text{если } x < -1. \end{cases}$

17. $y = \begin{cases} a+bx, & \text{если } x < 93; \\ b-ac, & \text{если } 93 \le x \le 120; \\ abx, & \text{если } x > 120. \end{cases}$

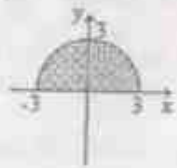
18. $y = \begin{cases} x^2-0.3, & \text{если } y < 3; \\ 0, & \text{если } 3 \le x \le 5; \\ x^2+1, & \text{если } x > 5. \end{cases}$

19. $y = \begin{cases} \sqrt{5x^2+5}, & \text{если } |x| < 2; \\ \dfrac{|x|}{\sqrt{5x^2+5}}, & \text{если } 2 \le |x| < 10; \\ 0, & \text{если } |x| \ge 10. \end{cases}$

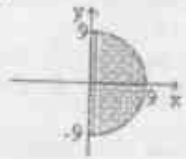
20. $y = \begin{cases} \sin(x), & \text{если } |x| < \dfrac{\pi}{2}; \\ \cos(x), & \text{если } \dfrac{\pi}{2} \le |x| \le \pi; \\ 0, & \text{если } |x| > \pi. \end{cases}$

VIII. Дана точка на плоскости с координатами (x, y). Составить программу, которая выдает одно из сообщений «Да», «Нет», «На границе» в зависимости от того, лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе. Области задаются графически следующим образом:

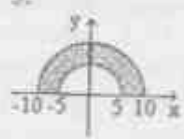
1.



2.



3.

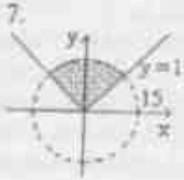


4.

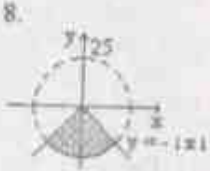
5.

6.

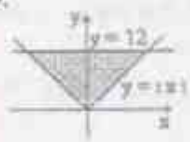
7.



8.

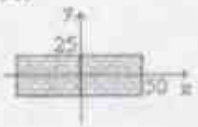


9.

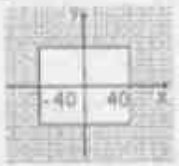


10.

11.

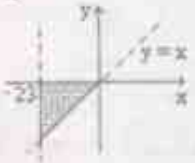


12.



13.

14.



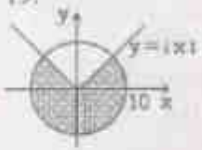
15.

16.

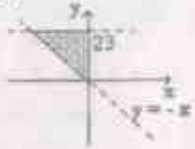
17.

18.

19.



20.



VIII. Составить программу.

1) Дан порядковый номер месяца, вывести на экран его название.
2) Дан порядковый номер дня недели, вывести на экран количество дней оставшихся до конца недели.
3) Дан порядковый номер месяца, вывести на экран количество месяцев оставшихся до конца года.
4) Дан порядковый номер дня месяца, вывести на экран количество дней оставшихся до конца месяца.
5) Дан номер масти m ($1 \le m \le 4$), определить название масти. Масти нумеруются: «пики» - 1, «трефы» - 2, «бубны» - 3, «червы» - 4.
6) Дан номер карты k ($6 \le k \le 14$), определить достоинство карты. Достоинства определяются по следующему правилу: «туз» - 14, «король» - 13, «дама» - 12, «валет» - 11, «десятка» - 10, …, «шестерка» - 6.
7) Дан номер масти m ($1 \le m \le 4$) и номер достоинства карты k ($6 \le k \le 14$). Определить полное название соответствующей карты в виде «дама пик», «шестерка бубен» и т.д.
8) С начала 1 января 1990 года по некоторый день прошло m месяцев, определить название месяца этого дня.
9) С начала 1 января 1990 года по некоторый день прошло m месяцев и n дней, определить название месяца этого дня.
10) Дано расписание приемных часов врача. Вывести на экран приемные часы врача в заданный день недели (расписание придумать самостоятельно).
11) Проведен тест, оцениваемый в целочисленных баллах от нуля до ста. Вывести на экран оценку тестируемого в зависимости от набранного количества баллов: от 90 до 100 – «отлично», от 70 до 89 – «хорошо», от 50 до 69 – «удовлетворительно», менее 50 – «неудовлетворительно».

12) Дан символ. Вывести на экран является ли он буквой, цифрой, знаком препинания, знаком арифметической операции, скобкой или служебным символом.
13) Даны два числа и арифметическая операция. Вывести на экран результат этой операции.
14) Дана буква русского алфавита. Вывести на экран является ли эта буква гласной, согласной или разделительным знаком.
15) Дан год. Вывести на экран название животного, символизирующего этот год по восточному календарю.
16) Дан возраст человека мужского пола в годах. Вывести на экран возрастную категорию: до года – «младенец», от года до 11 лет – «ребенок», от 12 до 15 лет – «подросток», от 16 до 25 лет – «юноша», от 26 до 70 лет – «мужчина», более 70 лет – «старик».
17) Дан пол человека: м – мужчина, ж – женщина. Вывести на экран возможные мужские и женские имена в зависимости от введенного пола.
18) Дан признак транспортного средства: а – автомобиль, в – велосипед, м - мотоцикл, с – самолет, п – поезд. Вывести на экран максимальную скорость транспортного средства в зависимости от введенного признака.
19) Дан номер телевизионного канала. Вывести на экран наиболее популярные программы заданного канала.
20) Дан признак геометрической фигуры на плоскости: к – круг, п – прямоугольник, т - треугольник. Вывести на экран периметр и площадь заданной фигуры (данные, необходимые для расчетов, запросить у пользователя).

2.5. Самостоятельная работа

Задача 1. Дана шахматная доска размером n×n клеток. Верхняя левая клетка доски черная и имеет номер (1, 1). Например, для n=4 шахматная таблица выглядит следующим образом:

1) для заданного значения n определить количество черных ячеек шахматной доски;
2) по номеру ячейки (k, m) определить ее цвет;
3) определить, являются ли ячейки с номерами (k1, m1) и (k2, m2) одного цвета;
4) определить, находится ли фигура, стоящая в ячейке с номером (k1, m1), под ударом второй фигуры, стоящей в ячейке с номером (k2, m2), при условии, что ход второй фигуры и ей является: a) пешка; b) слон; c) ладья; d) ферзь; e) конь.

Задача 2. Ближайшая к дому булочная работает с 7.00 до 19.00 и закрывается на перерыв с 13.00 до 15.00. Хлебный магазин, расположенный дальше, работает с 8.00 до 20.00 и имеет перерыв с 14.00 до 16.00. С 8.00 до 24.00 хлеб можно купить в гастрономе, который расположен дальше всех, но который работает без обеда. По времени на часах у Васи определите, что ему лучше сделать: сходить в булочную, дойти до хлебного магазина, съездить в гастроном или сидеть дома, так как везде закрыто.

Время вводится в виде действительного числа, в котором целая часть обозначает часы, а действительная часть – минуты. Например, 13.30 – 13 часов 30 минут.

Задачи 3. Задана дата в формате <день>.<месяц>.<год>. Определить:
1) сколько дней прошло с начала года; 2) сколько дней осталось до конца года;
3) дату предыдущего дня; 4) дату следующего дня.

3. ОПЕРАТОРЫ ЦИКЛА

Операторы цикла используются для многократного повторения некоторого фрагмента программы, который называется телом цикла. Циклы бывают арифметические и итерационные. Арифметический цикл – это такой цикл, в котором число повторений известно заранее. В ТР такой цикл обычно реализуется с помощью оператора *for*. В итерационных циклах число повторений заранее неизвестно, и выход из цикла производится в случае выполнения (или невыполнения) какого-то условия. Такие циклы в ТР реализуются с помощью операторов *while* и *repeat*.

3.1. Оператор цикла FOR

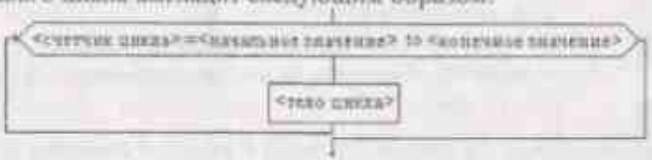
Оператор цикла *for* имеет следующую структуру:

for <счетчик цикла>:=<начальное значение> to <конечное значение> do <тело цикла>;

где *for, to, do* – зарезервированные слова (для, до, выполнить); *<счетчик цикла>* – счетчик или параметр цикла, который является переменной порядкового типа; *<начальное значение>, <конечное значение>* – выражения того же типа, что и счетчик; *<тело цикла>* – произвольный оператор Паскаля, простой или составной.

При выполнении оператора *for* вначале вычисляются начальное и конечное значения счетчика (это делается только один раз), а затем счетчику присваивается вычисленное начальное значение. После чего циклически повторяется следующая последовательность действий: если текущее значение счетчика больше конечного значения, то цикл завершает свою работу, иначе выполнятся тело цикла и значение счетчика увеличивается на 1.

Схема такого цикла выглядит следующим образом:



В качестве иллюстрации выполнения цикла *for* рассмотрим программу вывода на экран целых чисел из интервала от 1 до *n*.

```
program example3_1;
 var  i,n:integer;
begin
  writeln('Введите количество чисел'); readln(n)
  for i:=1 to n do      {с помощью цикла переменная i принимает все значения от 1 do n}
   write(' ',i);  {на каждом шаге выводим на экран текущее значение счетчика цикла}
end.
```

Рассмотрим по шагам выполнение данного цикла:

№ шага	*Значение счетчика*	*Результат (состояние экрана)*
1	i=1	1
2	i=2	1 2
3	i=3	1 2 3
…	…	…
n	i=n	1 2 3 4 … n

Если в качестве n ввести значение 5, то на экран будет выведено: 1 2 3 4 5.

Отметим важное обстоятельство: если начальное значение счетчика окажется меньше конечного значения, то цикл не выполнится ни разу. Например, если в качестве n ввести значение 0, то на экран ничего выведено не будет.

Существует и другая форма оператора цикла:

for <счетчик цикла>:=<начальное значение> downto <конечное значение> do
 <тело цикла>;

Замена зарезервированного слова *to* на слово *downto* означает, что после каждого выполнения тела цикла счетчик будет уменьшаться на 1, а выход из цикла произойдет тогда, когда текущее значение счетчика станет меньше конечного значения. При этом если начальное значение счетчика окажется меньше конечного значения, то цикл не выполнится ни разу.

Следующая модификация программы example3_1 приведет к тому, что при *n=5* на экран будут выведены числа 5 4 3 2 1.

```
program example3_2;
 var  i,n:integer;
begin
 writeln('Введите количество чисел'); readln(n);     {ввод с клавиатуры значения n}
 {с помощью цикла переменная i принимает все значения из интервала от n до 1}
 for i:=n downto 1 do
  write(' ',i);  {выводим на экран текущее значение счетчика цикла}
end.
```

3.2. Оператор цикла WHILE

Оператор *while* является оператором цикла с предусловием и записывается следующим образом:

while <условие> do <тело цикла>;

где *while, do* – зарезервированные слова (пока, делать); *<условие>* - выражение логического типа, истинность которого проверяется; *<тело цикла>* – произвольный оператор ТР, простой или составной.

При выполнении оператора *while* вначале проверяется условие, если оно оказывается истинным, то выполняется тело цикла, а затем опять проверяется условие. Как только условие окажется ложным, цикл завершит свою работу.

Схема такого цикла выглядит следующим образом:

- начало цикла *while*

-условие завершения цикла

- тело цикла

В качестве иллюстрации выполнения цикла *while* рассмотрим программу вывода на экран целых чисел из интервала от 1 до *n*.

```
program example3_3;
 var i,n:integer;
begin
 writeln('Введите количество чисел'); readln(n);      {ввод с клавиатуры значения n}
 i:=1; { переменная i принимает значение 1}
 while i<=n do        {пока i меньше или равно n}
  begin
   write(' ',i); i:=i+1;  {выводим на экран текущее значение i и увеличиваем его на 1}
  end;
end.
```

Заметим, что если условие окажется ложным при первой проверке, то тело цикла не выполнится ни разу. Например, если с клавиатуры в качестве n ввести значение 0, то на экран ничего не будет выведено.

Очень важно помнить о том, что если условие во время работы цикла не будет изменяться, то возможна ситуация зацикливания, т.е бесконечной работы цикла. Поэтому внутри тела должны находиться операторы, приводящие к изменению логического условия так, чтобы цикл мог корректно завершиться.

Подумайте, что будет выведено на экран при выполнении следующей программы:

```
program example3_4;
 var i,n:integer;
begin
 writeln('Введите количество чисел'); readln(n);
 i:= n;
 while i<=n do
   begin write(' ',i); i:=i-1; end;
end.
```

3.3. Оператор цикла REPEAT

Оператор *repeat* является оператором цикла с постусловием и записывается следующим образом:

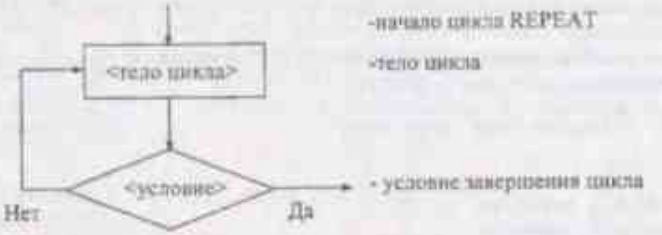
repeat <тело цикла> until <условие>;

где repeat, until – зарезервированные слова (повторять, до тех пор пока); *<тело цикла>* – произвольный оператор ТР, простой или составной (фактически *repeat* и *until* ограничивают тело цикла, поэтому если телом цикла является составной оператор, то структурные скобки *begin* и *end* ставить не надо); *<условие>* - выражение логического типа, истинность которого проверяется.

При выполнении оператора *repeat* вначале выполняется тело цикла, а затем проверяется условие. Если условие окажется истинным, то цикл завершит свою работу, а иначе тело цикла повторится еще раз.

Так как условие завершения цикла проверяется в конце цикла, то тело цикла выполняется хотя бы один раз. Однако, если условие всегда будет оставаться ложным, то возникнет ситуация зацикливания.

Схема данного цикла выглядит следующим образом:



-начало цикла REPEAT

-тело цикла

- условие завершения цикла

В качестве иллюстрации выполнения цикла *repeat* рассмотрим программу вывода на экран целых чисел из интервала от 1 до *n*.

```
program example3_5;
 var i,n:integer;
begin
 writeln('Введите количество чисел'); readln(n);    {ввод с клавиатуры значения n}
 i:=1; { переменная i принимает значение 1}
 repeat {повторяем}
  write(' ',i);  i:=i+1;{выводим на экран текущее значение i и увеличиваем его на единицу}
until i>n;      {до тех пор пока  i не станет больше n}
end.
```

Подумайте, что будет выведено на экран после выполнения следующей программы:

```
program example3_6;
 var i,n:integer;
begin
 writeln('Введите количество чисел'); readln(n);
 i:=1;
 repeat write(' ',i);    i:=i-1;  until i>n;
end.
```

3.4. Примеры использования операторов цикла при решении задач

Некоторые задачи можно решить с помощью любого циклического оператора. Вместе с тем, бывают ситуации, когда использование какого-то конкретного оператора цикла может оказаться более эффективным. Поэтому очень важно научиться выбирать оператор цикла в зависимости от задачи.

1. Написать программу, которая выводит на экран квадраты всех целых чисел от A до B (A и B целые числа, при этом A≤B).

Указания по решению задачи. Необходимо перебрать все целые числа из интервала от A до B. Эти числа представляют собой упорядоченную последовательность, в которой каждое число отличается от предыдущего на 1. Для решения данной задачи можно использовать любой оператор цикла, хотя короче будет выглядеть программа при использовании цикла for.

```
program example3_7;      {применение цикла for}
 var i, a, b:integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
```

```
{с помощью цикла переменная i последовательно принимает все значения от a до b}
for i:= a to b do
  write(' ',sqr(i));     {на каждом шаге цикла выводим на экран квадрат текущего }
end.                     {значения счетчика цикла}
```

```
program example3_8;      {применение цикла while}
 var  i, a, b :integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
 i:=a; { переменная i принимает значение a}
 while i<=b do           {пока i меньше или равно b}
  begin  write(' ',sqr(i));    {выводим на экран квадрат текущего значения i}
         i:=i+1; end;          {увеличиваем i на единицу}
end.
```

```
program example3_9; {применение цикла repeat}
 var  i, a, b:integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
 i:=a; { переменная i принимает значение a}
 repeat {повторяем}
   write(' ',sqr(i));  i:=i+1;   {выводим  на экран квадрат  i и увеличиваем значение i на 1}
 until i>b;                      {до тех пор пока  i не станет больше b}
end.
```

Все программы дадут нам следующий результат:

a	*b*	*состояние экрана*
2	5	4 9 25
-3	2	9 4 1 0 1 4

2. Написать программу, которая выводит на экран квадраты всех четных чисел из диапазона от A до B (A и B целые числа, при этом A≤B).

Указания по решению задачи. Из диапазона целых чисел от A до B необходимо выбрать только четные числа. Напомним, что четными называются числа, которые делятся на два без остатка. Кроме того, четные числа представляют собой упорядоченную последовательность, в которой каждое число отличается от предыдущего на 2. Попробуем решить эту задачу с помощью каждого оператора цикла.

```
program example3_10;     {применение цикла for}
 var  i, a, b:integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
 for i:= a to b do       {с помощью цикла переменная i перебирает все значения от a до b}
  if i mod 2 =0          {если i делится на два без остатка,}
   then  write(' ',i*i);  { то i – является четным числом и мы выводим его квадрат на экран }
end.
```

```
program example3_11;     {применение цикла while}
 var  i, a, b :integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
 if a mod 2 =0           {если a является четным числом, }
```



```
  then i:=a              {то в качестве i берем значение  a}
  else i:= a+1;          {иначе a является нечетным числом и в качестве i мы берем первое}
                         {четное число, которое следует за a}
 while i<=b do           {пока i меньше или равно b}
  begin  write(' ', i*i);      {выводим на экран квадрат  i и увеличиваем значение i на 2}
         i:=i+2;  end;
end.
```

```
program example3_12; {применение цикла repeat}
 var  i, a, b:integer;
begin
 writeln('Введите число A:'); readln(a);
 writeln('Введите число B:'); readln(b);
 if a mod 2 =0           {если a является четным числом, }
  then i:=a              {то в качестве i берем значение  a}
  else i:= a+1;          {иначе a является нечетным числом и в качестве i мы берем первое}
                         {четное число, которое следует за a}
 repeat {повторяем}
   write(' ', i*i);  i:=i+2;{выводим на экран квадрат  i  и увеличиваем его значение на 2}
until i>b; {до тех пор пока  i не станет больше b}
end.
```

Все программы дадут нам следующий результат:

a	*b*	*состояние экрана*
2	9	4 16 36 64
-3	6	4 0 4 16 36

Замечание. Хотя в данном случае все программы решают одну и ту же задачу, но применение операторов цикла *while* и *repeat* будет более эффективным, чем применение цикла *for*. Это связано с тем, что в циклах *while* и *repeat* значение переменной *i* можно увеличивать (уменьшать) произвольным образом. В нашем случае мы увеличивали на 2, чтобы из указанного диапазона перебирать только четные числа. А в цикле *for... to* значение счетчика может увеличиваться только на 1. Поэтому в программе example3_10 мы перебирали все числа, и каждое из них проверяли на четность.

3. Написать программу, которая выводит на экран целые числа 17, 24, …, 45 в обратном порядке.

Указания по решению задачи. По условию задачи необходимо вывести на экран числа 45 38 31 24 17. Указанные числа представляют собой убывающую арифметическую прогрессию, в которой каждое число отличается от предыдущего на 7. Поэтому данную задачу можно легко решить с помощью операторов цикла *while* и *repeat*.

```
program example3_13;     {применение цикла while}
 var  i:integer;
begin
 i:=45; { переменная i принимает значение первого числа из указанного диапазона}
 while i>=17 do          {пока i больше или равно 17}
  begin  write(' ', i); i:=i-7; end; {выводим на экран  значение i и уменьшаем его на 7}
end.
```

```
program example3_14; {применение цикла repeat}
 var  i:integer;
begin
 i:=45; { переменная i принимает значение первого числа из указанного диапазона}
 repeat {повторяем}
  write(' ', i); i:=i-7;  {выводим на экран  значение i и уменьшаем его на 7}
```

```
until i<17;{до тех пор пока  i не станет меньше 17}
end.
```

Данную задачу так же можно решить с помощью цикла *for...downto*, если внимательно проанализировать указанные числа: все числа последовательности 45 38 31 24 17 при делении на 7 дают остаток 3.

```
program example3_15;    {применение цикла for}
 var  i:integer;
begin
for i:= 45 downto 17do     { переменная i перебирает все значения от 45 до17}
  if i mod 7 =3        {если i делится на 7 с остатком 3,}
   then  write( ',i);   { мы выводим i на экран }
end.
```

3.5. Упражнения

I. Объяснить, что будет напечатано каждой программой:

```
1) program e1;
    var x: byte;
   begin
    for x:=1 to 5 do
     writeln(x+1);
   end.
```

```
2) program e2;
    var x, y: byte;
   begin
    y:=0;
    for x:=1 to 5 do y:=y+x;
    write(y);
   end.
```

```
3) program e3;
    var x, y: byte;
   begin
    y:=0;
    for x:=1 to 5 do begin
     y:=y+x; write(y); end;
    end.
```

```
4) program e4;
    var a: integer;
   begin
    a:=0;
    while a<20 do
     begin write(a, ' ');
           a:=a+5;  end;
   end.
```

```
5) program e5;
    var a: integer;
   begin
    a:=10;
    while a<0 do
     begin write(a, ' ');
           a:=a-3; end;
   end.
```

```
6) program e6;
    var a, b: byte;
   begin
    a:=1; b:=10;
    while a<=b do
     begin a:=a+1; b:=b-2; end;
    write(a+b);
   end.
```

```
7) program e7;
    var a, b, i:integer;
   begin
    a:=3; b:=15; i:=1;
    while (i>=a) and (i<=b) do
     begin write(i, ' ');
           i:=2*i; end;
   end.
```

```
8) program e8;
    var a, b, i: integer;
   begin
    a:=3; b:=15; i:=1;
    while (i<=100) or (i<=b) do
     begin write(i, ' ');
           i:=2*i; end;
   end.
```

```
9) program e9;
    var n: integer;
   begin
    n:=0;
    repeat
     write(n, ' '); n:=n+4;
    until n>=15;
   end.
```

```
10) program e10;
     var n: integer;
    begin
     n:=1;
     repeat
      write(n, ' ');
      n:=n-2;
     until n<0;
    end.
```

```
11) program e11;
     var n, m: integer;
    begin
     n:=1; m:=0;
     repeat
      m:=m+n;  n:=n+1;
     until (n>=10) or (m>=20);
     write(m);
    end.
```

```
12) program e12;
     var n, m: integer;
    begin
     n:=1; m:=0;
     repeat
      m:=m+n; n:=n+1;
     until (n>=10) and (m>=20);
     write(m);
    end.
```

II. Найти и объяснить ошибки в каждом фрагменте программы:


```
1) for i:=1 to n/2 do
    writeln(i);
```

```
2) for i:=10 to 1 do
    writeln(i);
```

```
3) for c:=3 to 7 do;
    writeln(i);
```

```
4) a:=1;
   while a<10 do
    write (a);
    a:=a+1;
```

```
5) a:=10;
   while a>1 do;
    begin
     write (a);  a:=a-1;
    end;
```

```
6) a:=10; i:=1;
   while i<=a and i mod 2 =1 do
    i:=i+2;
    write(i);
```

```
7) x:=100;
   repeat
    write(x);
    x:=x div 2;
  until x=0 do;
```

```
8) x:=2;
   repeat
    begin write(x);
          x:=x *x; end;
  until x>100 do;
```

```
9) read(x);
   repeat
    x:=x +x;
  until x<-100 or x>100;
  write(x);
```

II. Вывести на экран:

Замечание. Решите каждую задачу тремя способами: используя операторы цикла *for*, *while* и *repeat*.

1) целые числа 1, 3, 5, …, 21 в строчку через пробел;
2) целые числа 10, 12, 14, …, 60 в обратном порядке в столбик;
3) числа следующим образом:

```
10 10.4
11 11.4
...
25 25.4
```

4) числа следующим образом:

```
25 25.5 24.8
26 26.5 25.8
...
35 35.5 34.8
```

5) таблицу соответствия между весом в фунтах и весом в килограммах для значений 1, 2, 3, …, 10 фунтов (1 фунтов = 453г);
6) таблицу перевода 5, 10, 15, …, 120 долларов США в рубли по текущему курсу (значение курса вводится с клавиатуры);
7) таблицу стоимости для 10, 20, 30,…, 100 штук товара, при условии что одна штука товара стоит *x* руб (значение *x* водится с клавиатуры);
8) таблицу перевода расстояний в дюймах в сантиметры для значений 2, 4, 6, …, 12 дюймов (1 дюйм = 25.4 мм);
9) кубы всех целых чисел из диапазона от A до B (A≤B) в обратном порядке;
10) все целые числа из диапазона от A до B (A≤B), оканчивающиеся на цифру X;
11) все целые числа из диапазона от A до B (A≤B), оканчивающиеся на цифру X или У;
12) все целые числа из диапазона от A до B (A≤B), оканчивающиеся на любую четную цифру;
13) только положительные целые числа из диапазона от A до B (A≤B);
14) все целые числа из диапазона от A до B, кратные трем (A≤B);
15) все четные числа из диапазона от A до B, кратные трем (A≤B);
16) только отрицательные четные числа из диапазона от A до B (A≤B);
17) все двухзначные числа, в записи которых все цифры разные;
18) все двухзначные числа, в которых старшая цифра отличается от младшей не больше чем на 1;
19) все трехзначные числа, которые начинаются и заканчиваются на одну и ту же цифру;
20) все трехзначные числа, в которых хотя бы две цифры повторяются.

3.6. Самостоятельная работа

Задача 1. Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведенных в n-ную степень, равна самому числу. Например, $153=1^3+5^3+3^3$. Найти все трехзначные числа Армстронга.

Задача 2. Стороны прямоугольника заданы натуральными числами n и m. Найти количество квадратов (стороны которых выражены натуральными числами), на которые можно разрезать данный прямоугольник, если от него каждый раз отрезать квадрат: 1) наименьшей площади; 2) наибольшей площади.

4. ВЛОЖЕННЫЕ ЦИКЛЫ

4.1. Примеры использования вложенных операторов цикла

Циклы могут быть не только простые, но вложенные (кратные, циклы в цикле). Вложенными могут быть циклы любых типов: *for*, *while*, *repeat*. Структура вложенных циклов на примере оператора цикла *for* приведена ниже:

```
                                  Уровни вложенности
for i:=1 to ik do begin ...
  for j:=1 to jk do begin ...
    for k:=1 to kk do begin ...      3
                  end; ...           2
                end; ...             1
              end;
```

Каждый внутренний цикл должен быть полностью вложен во все внешние циклы. «Пересечения» не допускаются. Рассмотрим несколько примеров составления программ с использованием вложенных циклов.

1. Напечатать числа в виде следующей таблицы:

```
2  2  2  2  2
2  2  2  2  2
2  2  2  2  2
2  2  2  2  2
```

Указания по решению задачи. Данная таблица состоит из четырех строчек, в каждой из которых число два напечатано пять раз. Для решения поставленной задачи можем использовать вложенные операторы *for*. Внутренний оператор печатает строчку из пяти двоек, а внешний оператор организует повторение печати этой строчки четыре раза.

```
program example4_1;
  var i, j: byte;
begin
  for i:= 1 to 4 do     {начало внешнего цикла}
    begin
      for j:= 1 to 5 do write(2, ' ');   {внутренний цикл}
      writeln;          {перевод курсора на новую строчку}
    end;                {конец внешнего цикла}
end.
```


2. Напечатать числа в виде следующей таблицы:

```
10  9  8  ...  2
10  9  8  ...  2
10  9  8  ...  2
```

Указания по решению задачи. Данная таблица состоит из трех строчек, в каждой из которых печатаются целые числа от 2 до 10 в обратном порядке. Для решения поставленной задачи можем использовать вложенные операторы *for*. Внутренний оператор печатает строчку целых чисел от 2 до 10 в обратном порядке, а внешний оператор организует повторение печати этой строчки три раза.

```
program example4_2;
  var i, j: byte;
begin
  for i:= 1 to 3 do          {начало внешнего цикла}
    begin
      for j:= 10 downto 2 do write(j, ' ');   {внутренний цикл}
      writeln;               {перевод курсора на новую строчку}
    end;                     {конец внешнего цикла}
end.
```

3. Напечатать числа в виде следующей таблицы:

```
1
1  3
1  3  5
1  3  5  7
1  3  5  7  9
```

Указания по решению задачи. В данном случае таблица состоит из пяти строчек, в каждой из которых печатаются только нечетные числа. Причем последнее нечетное число в строчке зависит от номера строки. Эта зависимость выражается через формулу $k = 2i-1$ (зависимость проверить самостоятельно), где k – последнее число в строке, i – номер текущей строки. Для решения поставленной задачи можем использовать оператор *while*, вложенный в оператор *for*. Внешний оператор следит за номером текущей строки i, а внутренний оператор будет печатать нечетные числа из диапазона от 1 до $2i-1$.

```
program example4_3;
  var i, j: byte;
begin
  for i:= 1 to 5 do          {начало внешнего цикла}
    begin
      j:=1;
      while j<=2*i-1 do          {внутренний цикл}
        begin write(j, ' '); j:=j+2; end;
    writeln;                 {перевод курсора на новую строчку}
    end;                     {конец внешнего цикла}
end.
```

4. Напечатать числа в виде следующей таблицы:

```
1
3
2  2
4  4
3  3  3
5  5  5
4  4  4  4
6  6  6  6
```

Указания по решению задачи. Исходную таблицу можно схематично разбить на четыре группы строк:

В каждой группе две строки, для элементов которых выполняется следующее правило: в первой строчке i-группы напечатано число i, причем i-раз, а во второй строчке i-группы напечатано число $i+2$ i-раз. Для решения поставленной задачи можем использовать два последовательных оператора *for*, вложенных в оператор *for*. Внешний оператор следит за номером текущей группы i, первый вложенный цикл i раз печатает число i, второй вложенный цикл i раз печатает число $i+2$.

```
program example4_4;
 var i, j: byte;
begin
 for i:= 1 to 4 do              {начало внешнего цикла}
  begin
   for j:=1 to i do write(i, ' ');     {первый вложенный цикл}
   writeln;                            {переход курсора на новую строчку}
   for j:=1 to i do write(i+2, ' ');   {второй вложенный цикл}
   writeln;                            {переход курсора на новую строчку}
  end;                           {конец внешнего цикла}
end.
```

4.2.Упражнения

I. Объяснить, что будет напечатано каждой программой:

1)
```
program e1;
 var i, j: byte;
begin
 for i:=1 to 5 do
  for j:=1 to 5 do writeln(i+j, ' ');
end.
```

2)
```
program e2;
 var i, j: byte;
begin
 for i:=1 to 4 do
  for j:=1 to 3 do write(i*j);
end.
```

3)
```
program e3;
 var i, j: byte;
begin
 for i:=1 to 3 do
  begin
   for j:=2*i downto 1 do write(j);
   writeln;
  end;
end.
```

4)
```
program e4;
 var a, b: integer;
begin
 a:=0;
 while a<10 do
  begin
   b:=a;
   while b>0 do
    begin write(b, ' '); b:=b-2; end;
   writeln; a:=a+2;
  end;
end.
```


5)
```
program e5;
 var a, b: integer;
begin
 a:=0;
 while a<10 do
  begin
   b:=a;
   while b>0 do
    begin write(b, ' '); b:=b-2; end;
   writeln; a:=a+2;
  end;
end.
```

6)
```
program e6;
 var a, b: integer;
begin
 a:=0;
 while a<10 do
  begin
   write(a,' '); a:=a+2;
  end;
 b:=a;
 while b>0 do
  begin write(b, ' '); b:=b-2; end;
end.
```

II. Найти и объяснить ошибки в каждом фрагменте программы:

1)
```
for i:=1 to 5 do
 for i:= 3 to 6 do
  write(i);
```

2)
```
for i:=1 to 5 do;
 for j:= 1 to 3 do
  write(i);
```

3)
```
for i:=1 to 5 do
 begin
  for j:= 1 to 3 do write(i);
  writeln; end;
 end;
```

4)
```
a:=0;
while a<10 do
 begin
  b:=a;
  while b<10 do
   write(b, ' '); b:=b+3;
  writeln; a:=a+2;
end;
```

5)
```
a:=0;
while a<10 do
 begin
  b:=a;
  repeat
   write(b, ' '); b:=b+1;
 end;
 writeln; a:=2*a;
 until b>10;
```

6)
```
a:= 1;
repeat
 b:=a;
 repeat
  a:=a+1; write(a, ' ');
 until b>5;
 b:=b+1;
until a>5;
```

III. Вывести на экран числа в виде следующей таблицы:

1)
```
5  5  5  5  5  5
5  5  5  5  5  5
5  5  5  5  5  5
5  5  5  5  5  5
```

2)
```
1  2  3  ...  10
1  2  3  ...  10
1  2  3  ...  10
1  2  3  ...  10
```

3)
```
-10  -9  -8  ...  12
-10  -9  -8  ...  12
-10  -9  -8  ...  12
-10  -9  -8  ...  12
-10  -9  -8  ...  12
```

4)
```
41  42  43  ...  50
51  52  53  ...  60
61  62  63  ...  70
...
71  72  73  ...  80
```

5)
```
5
5  5
5  5  5
5  5  5  5
5  5  5  5  5
```

6)
```
1  1  1  1  1
1  1  1  1
1  1  1
1  1
1
```

7)
```
1
2  2
3  3  3
4  4  4  4
5  5  5  5  5
```

8)
```
6  6  6  6  6
7  7  7  7
8  8  8
9  9
10
```

9)
```
7
6  6
5  5  5
4  4  4  4
3  3  3  3  3
```

10)
```
8  8  8  8  8
7  7  7  7
6  6  6
5  5
4
```

11)
```
1
1  2
1  2  3
1  2  3  4
1  2  3  4  5
```

12)
```
1
2  1
3  2  1
4  3  2  1
5  4  3  2  1
```

13)
```
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0
```

14)
```
4 3 2 1 0
3 2 1 0
2 1 0
1 0
0
```

15)
```
1
0
2 2
0 0
3 3 3
0 0 0
4 4 4 4
0 0 0 0
5 5 5 5 5
0 0 0 0 0
```

16)
```
8
7
7 7
6 6
6 6 6
5 5 5
5 5 5 5
4 4 4 4
```

17)
```
1
6
2 2
7 7
3 3 3
8 8 8
4 4 4 4
9 9 9 9
```

18)
```
9
4
8 8
3 3
7 7 7
2 2 2 2
6 6 6 6 6
1 1 1 1 1
```

19)
```
3
0
2 3
9 0
2 2 3
8 9 0
2 2 2 3
7 8 9 0
2 2 2 2 3
6 7 8 9 0
```

20)
```
2 2 2 2 2
3 4 5 6 7
2 2 2 2
2 3 4 5
2 2 2
1 2 3
2 2
0 1
2
-1
```

4.3. Самостоятельная работа

Задача 1. Дано натуральное число n. Определить количество решений неравенства $x^2+y^2<n$ в натуральных числах.

Задача 2. Дано натуральное число n. Найти все тройки натуральных чисел x, y, z (если они существуют) такие, что $x^2+y^2+z^2=n$.

5.РЕКУРРЕНТНЫЕ СООТНОШЕНИЯ

5.1. Вычисление членов рекуррентной последовательности

Пусть $a_1, a_2, \ldots, a_n$ - произвольная числовая последовательность. Рекуррентным соотношением называется такое соотношение между членами последовательности, в котором каждый следующий член выражается через несколько предыдущих, т.е $a_k = f(a_{k-1}, a_{k-2}, \ldots, a_{k-l}),\ k > l$ (1).

Последовательность задана рекуррентно, если для нее определено рекуррентное соотношение вида (1) и заданы первые l ее членов.

Самым простым примером рекуррентной последовательности является арифметическая прогрессия. Рекуррентное соотношение для нее записывается в виде:

$a_k = a_{k-1} + d$, где d – разность прогрессии. Из школьного курса алгебры вы помните, что, зная первый элемент и разность прогрессии, и, используя данное рекуррентное соотношение, можно последовательно вычислить все остальные члены прогрессии.

Рассмотрим пример программы, в которой вычисляются первые n членов арифметической прогрессии при условии, что $a_1 = \frac{1}{2}$ и $d = \frac{1}{4}$.

```
program example5_1;
 var a, d:real; n, i: byte;
begin
 a:=0.5; d:=0.25;  {задали первый член последовательности и разность прогрессии}
 write('n='); readln(n); {ввели количество членов последовательности}
 writeln('1 элемент: ', a:5:2);      {вывели первый член последовательности}
 for i:=2 to n do      {организуем вычисление 2, 3, ... ,n члена последовательности}
  begin
    a:=a+d;          {для этого прибавляем к предыдущему члену значение d}
    writeln(i, 'элемент: ', a:5:2);   {и выводим новое значение a на экран}
  end;
end.
```

Результат работы программы:

n	*состояние экрана*
5	1 элемент: 0.50
	2 элемент: 0.75
	3 элемент: 1.00
	4 элемент: 1.25
	5 элемент: 1.5

Другим примером рекуррентной последовательности является геометрическая прогрессия. Рекуррентное соотношение для нее записывается в виде: $b_k = b_{k-1} * q$, где q – знаменатель прогрессии. Рассмотрим пример программы, в которой вычисляются первые n членов арифметической прогрессии при условии, что $b_1=1$, $q=2$.

```
program example5_2;
 var b, q, n, i: word;
begin
 b:=1; q:=2;  {задали первый член последовательности и знаменатель прогрессии}
 write('n='); readln(n); {ввели количество членов последовательности}
 writeln('1 элемент: ', b);  {вывели первый член последовательности}
 for i:=2 to n do      {организуем вычисление 2, 3, ... ,n члена последовательности}
  begin
    b:=b*q;          {для этого умножаем предыдущее значение b на q}
    writeln(i, 'элемент: ', b);{и выводим новое значение b на экран}
  end;
end.
```

Результат работы программы:

n	*состояние экрана*
5	1 элемент: 1
	2 элемент: 2
	3 элемент: 4
	4 элемент: 8
	5 элемент: 16

В арифметической и в геометрической прогрессиях каждый член последовательности зависит только от одного предыдущего значения. Более сложная зависимость представлена в последовательности Фибоначчи: $a_1 = a_2 = 1$, $a_n = a_{n-1} + a_{n-2}$. В

этом случае каждый член последовательности зависит от значений двух предыдущих членов. Рассмотрим пример программы, в которой вычисляются первые n членов последовательности Фибоначчи.

```
program example5_3;
 var  a1, a2, a, n, i:integer;
begin
 a1:=1; a2:=1; {задали первый и второй члены последовательности Фибоначчи }
 write('n='); readln(n);      {ввели количество членов последовательности}
 writeln('1 элемент: ', a1); writeln('2 элемент: ', a2); {вывели известные члены}
 {Организуем цикл для вычисления членов последовательности с номерами 3, 4,..., n.
  При этом в переменной a1 будет храниться значение члена последовательности с
  номером i-2, в переменной a2 -  члена с номером i-1, переменная a будет использоваться для вычисления члена с номером i. }
 for i:=3 to n do
   begin
   {по рекуррентному соотношению вычисляем член последовательности с номером i
   и выводим его значение на экран}
   a:=a1+a2;        writeln(i, ' элемент: ', a);
   {выполняем рекуррентный пересчет для следующего шага цикла}
   a1:=a2;   {в элемент с номером i-2 записываем значение элемента с номером i-1}
   a2:=a;    {в элемент с номером i-1 записываем значение элемента с номером i}
 end;
end.
```

Результат работы программы:

n	*состояние экрана*
5	1 элемент: 1 2 элемент: 1 3 элемент: 2 4 элемент: 3 5 элемент: 5

В рассмотренных случаях мы выводили на экран значения первых n элементов рекуррентной последовательности. Иногда нам бывает необходимо найти только значение n-ного элемента последовательности. Для этого в программе нужно исключить вывод значения на каждом шаге цикла. В качестве примера рассмотрим программу, в которой вычисляется n-ный элемент последовательности, заданной следующим образом: $b_1 = 1, \quad b_2 = 2, \quad b_n = \frac{b_{n-1} - b_{n-2}}{(n-1)^2}$.

```
program example5_4;
 var  b1, b2, b: real;
      n, i: byte;
begin
 b1:=1; b2:=2; {задали первый и второй элементы последовательности}
 write('n='); readln(n);      {ввели количество элементов последовательности}
 {Организуем цикл для вычисления элементов с номерами 3, 4,..., n. При этом в переменной b1 будет храниться значение элемента последовательности с номером i-2, в
  переменной b2 -  элемента с номером i-1, переменная b будет использоваться для вычисления элемента с номером i. }
 for i:=3 to n do
   begin
   {по рекуррентному соотношению вычисляем  i-тый элемент последовательности}
   b:=(b1-b2)/sqr(i-1);
   {выполняем рекуррентный пересчет для следующего шага цикла}
   b1:=b2;   b2:=b;
 end;
 writeln(n, ' элемент: ', b:6:3);   {выводим значение b на экран}
end.
```

Результат работы программы:

n	*состояние экрана*
5	5 элемент: -0.031

5.2. Упражнение

I. Объяснить, что будет напечатано каждой программой:

1)
```
program e1;
 var a, d, n, i: integer;
begin
 a:=-2; d:=3;  n=6;  writeln('1 элемент: ', a);
 for i:=2 to n do
  begin a:=a+d;
        writeln(i, 'элемент: ', a); end;
end.
```

2)
```
program e2;
 var b, q: real;
     n, i: byte;
 begin
  b:=8; q:=0.5;  n=4;
  for i:=2 to n do b:=b*q;
  writeln(n, 'элемент: ', b);
 end.
```

3)
```
program e3;
 var  a1, a2, a, n, i: integer;
begin
 a1:=10; a2:=7; n:=5;
 write(a1, ' ', a2, ' ');
 for i:=3 to n do
  begin  a:=2*a1-a2;  write(a, ' ');
         a1:=a2;   a2:=a; end;
end.
```

4)
```
program e4;
 var  b1, b2, b: real;
      n, i: byte;
begin
 b1:=1; b2:=2; n:=5;
 for i:=3 to n do
  begin b:=b2+b1/i;  b1:=b2;   b2:=b; end;
 writeln(b:6:3);
end.
```

II.Объясните и исправьте ошибки в каждом фрагменте программы:

при условии, что нужно выводить каждый элемент последовательности

1) $a_1 = 3, \quad d = -3 \quad a_n = a_{n-1} - d, \quad n = 6$

```
a:=3; d:=3;  n=6;  writeln('1 элемент: ', a);
for i:=2 to n do
 begin a:=a+d;
       writeln(i, 'элемент: ', a); end;
```

2) $b_1 = 4, \quad q = -2 \quad b_n = b_{n-1} / d, \quad n = 4$

```
b:=4; q:=-2;  n=4;
for i:=2 to n do b:=b*q;
writeln(n, 'элемент: ', b);
end;
```

при условии, что нужно выводить n-ный элемент последовательности

3) $a_1 = 1, \quad a_2 = 2 \quad a_n = 2a_{n-1} + a_{n-2}, \quad n = 6$

```
a1:=1; a2:=2; n:=6;  write(a1, ' ', a2, ' ');
for i:=3 to n do
 begin
  a:=2*a1+a2;   write(a, ' ');
  a2:=a;   a1:=a2;
end;
```

4) $b_1 = 4, \quad b_2 = 2 \quad b_n = n(b_{n-1} + b_{n-2}), \quad n = 7$

```
b1:=4; b2:=2; n:=7;
for i:=2 to n do
 begin
  b:=n*(b2+b);  b1:=b2;  b2:=b;
 end;
writeln(b);
```

III. Написать программу, вычисляющую первые n элементов заданной последовательности:

1. $b_1 = 9, b_n = 0.1b_{n-1} + 10$;

2. $b_1 = -1, b_n = 9 - 2b_{n-1}$;

3. $b_1 = 1, b_n = 0.2b_{n-1}^4 + 1$;

4. $b_1 = 4.7, b_n = \sin(b_{n-1}) + \pi$;;

5. $b_1=0.1, b_n=\frac{1}{6}(0.05+b_{n-1}^3)$;

6. $b_1=2, b_n=0.5(\frac{1}{b_{n-1}}+b_{n-1})$

7. $b_1=5,\ b_n=(-1)^n b_{n-1}-8$;

8. $b_1=-1,\ b_2=1,\ b_n=3b_{n-1}-2b_{n-2}$;

9. $b_1=-10,\ b_2=2,\ b_n=|b_{n-2}|-6b_{n-1}$;

10. $b_1=2,\ b_2=4,\ b_n=6b_{n-1}-b_{n-2}$;

11. $b_1=5,\ b_n=\frac{b_{n-1}}{n^2+n+1}$;

12. $b_1=0.5,\ b_2=0.2,\ b_{n+1}=b_n^2+\frac{b_{n-1}}{n}$;

13. $b_1=1,\ b_n=\frac{1}{4}\left(3b_{n-1}+\frac{1}{3b_{n-1}}\right)$;

14. $b_1=2,\ b_2=1,\ b_n=\frac{2}{3}b_{n-2}-\frac{1}{3}b_{n-1}^2$;

15. $b_1=1,\ b_2=2,\ b_n=\frac{b_{n-2}}{4}+\frac{5}{b_{n-1}^2}$;

16. $b_1=1,\ b_2=2,\ b_n=\frac{nb_{n-2}-b_{n-1}}{n+1}$;

17. $b_1=4,\ b_2=2,\ b_n=\frac{b_{n-2}}{n}+\frac{n^2}{b_{n-1}}$

18. $b_1=100,\ b_{2n}=b_{2n-1}/10,\ b_{2n+1}=b_{2n}+10$;

19. $b_1=0,\ b_{2n}=b_{2n-1}+3,\ b_{2n+1}=2b_{2n}$;

20. $b_1=1,\ b_2=5\ \ b_{2n}=b_{2n-1}+b_{2n-2},\ b_{2n+1}=b_{2n}-b_{2n-1}$.

5.3. Самостоятельная работа

Задача 1. Сидоров открыл счет в банке 1 января 2007 года на сумму 20000 рублей под 14% годовых. Начисление процентов происходит первого числа каждого месяца, начисленная сумма добавляется к сумме вклада. Распечатайте состояние счета Сидорова по месяцам 2007 года

Задача 2. Петя решил заниматься спортом. Теперь каждый день он подтягивается на турнике. Через сколько дней он будет подтягиваться не меньше N раз в день, при условии, что в первый день он подтянулся 1 раз, а каждый следующий день собирается подтягиваться в два раза больше чем в предыдущий день?

6. ВЫЧИСЛЕНИЕ СУММ И ПРОИЗВЕДЕНИЙ

6.1. Вычисление конечных сумм и произведений

Решение многих задач связано с нахождением суммы или произведения элементов заданной последовательности. В данном разделе мы рассмотрим основные приемы вычисления конечных сумм и произведений.

Пусть $u_1(x),u_2(x),\ldots,u_n(x)$ - произвольная последовательность n функций. Будем рассматривать конечную сумму вида $u_1(x)+u_2(x)+\ldots+u_n(x)$. Такую сумму можно записать более компактно, используя следующее обозначение: $u_1(x)+u_2(x)+\ldots+u_n(x)=\sum_{i=1}^{n}u_i(x)$. При $n\le 0$ значение суммы равно 0.

В дальнейшем будем также использовать сокращенную запись для конечного произведения данной последовательности, которая выглядит следующим образом: $u_1(x)\cdot u_2(x)\cdot\ldots\cdot u_n(x)=\prod_{i=1}^{n}u_i(x)$.

1. Написать программу, которая подсчитывает сумму натуральных чисел от 1 до n (n≥1).

Указания по решению задачи. Пусть s_n - сумма натуральных чисел от 1 до n. Тогда s_n=1+2+…+(n-1)+n=(1+2+…+(n-1))+n=s_{n-1}+n, s_0=0. Мы пришли к рекуррентному соотношению s_0=0, s_n=s_{n-1}+n, которым мы можем воспользоваться для подсчета суммы. При этом формула s_n=s_{n-1}+n говорит о том, что значение суммы на n-ном шаге равна сумме, полученной на предыдущем шаге, плюс очередное слагаемое.

```
program example6_1;
 var i,n,s: integer;
begin
 writeln('Введите количество чисел'); readln(n);
 s:=0;
 for i:=1 to n do      {выполняем n шагов и на каждом i-том шаге}
   s:=s+i;             {используем полученное рекуррентное соотношение}
 writeln('s=',s);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Результат (значение s)*
1	i=1	0+1=1
2	i=2	1+2=3
3	i=3	1+2+3=6
…	…	
n	i=n	1+2+3+4+ … +n

2. Написать программу, которая считает x^n для вещественного x и натурального n.

Указание по решению задачи. Из свойства степенной функции (x^0=1, x^n=x^{n-1}*x) следует, что ее можно вычислять, используя рекуррентное соотношение b_0=1, b_n=b_{n-1}*x.

```
program example6_2;
 var i,n:integer;
     x, b:real;
begin
 writeln('Введите показатель степени'); readln(n);
 writeln('Введите основание степени'); readln(x);
 b:=1;
 for i:=1 to n do b:=b*x;
 writeln(x, 'в степени ', n, '= ', b);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Результат (значение b)*
1	i=1	1·x
2	i=2	x·x=x^2
3	i=3	x^2·x=x^3
…	…	…
n	i=n	x^{n-1}·x=x^n

3. Написать программу для подсчета суммы: -sinx+sin2x-sin3x+…+$(-1)^n$sinnx (n≥1).

Указания по решению задачи. Если пронумеровать слагаемые, начиная с номера 1, то мы увидим закономерность: знак минус ставится перед слагаемыми с нечетными номерами, а знак плюс – с четными, при этом сумму можно выразить рекуррентным соотношением: $s_0=0$, $s_n=s_{n-1}+(-1)^n \sin(n*x)$.

```
program example6_3;
 var i,n:integer;
    x,s:real;
begin
 writeln('Введите n'); readln(n);
 writeln('Введите x'); readln(x);
 s:=0;
 for i:=1 to n do
  if odd(i) then s:=s-sin(i*x) else  s:=s+sin(i*x);
 writeln('s=',s);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Результат (значение s)*
1	i=1	0-sinx=sinx
2	i=2	-sinx+sin2x
3	i=3	-sinx+ sin2x-sin3x
…	…	
n	i=n	-sinx+ sin2x-sin3x+…+(-1)sinnx

Эту же задачу можно решить без использования условного оператора *if*. Для этого нужно ввести вспомогательную переменную, например, *singl*, которая будет использоваться для хранения знака очередного слагаемого. Т.к. первое слагаемое отрицательно, то начальное значение переменной *singl* равно -1, а затем после каждой итерации цикла значение переменной *singl* будем заменять на противоположное с помощью унарного минуса. В этом случае рекуррентное соотношение можно преобразовать к виду $s_0=0$, $singl_0 = -1$, $s_n=s_{n-1}+singl*\sin(n*x)$, $singl_n=-singl_{n-1}$.

```
program example6_4;
 var i,n, singl:integer;
    x,s:real;
begin
 writeln('Введите n'); readln(n);
 writeln('Введите x'); readln(x);
 s:=0; singl:=-1;
 for i:=1 to n do
  begin s:=s+singl*sin(i*x);      singl:=-singl; end   (1)
writeln('s=',s);
end.
```

Подумайте, что произойдет, если поменять местами операторы присваивания в строке 1.

4. Написать программу для подсчета суммы $S_n=\frac{\cos x}{1}+\frac{\cos x+\cos 2x}{2}+\frac{\cos x+cox2x+\cos 3x}{3}+\ldots+\frac{\cos x+\ldots+\cos nx}{n}$, где x – вещественное число, n – натуральное число.

Указания по решению задачи. Если пронумеровать слагаемые, начиная с 1, то мы увидим, что номер слагаемого совпадает со значением знаменателя. Рассмотрим

каждый числитель отдельно: $b_1=\cos x$, $b_2=\cos x+\cos 2x$, $b_3=\cos x+\cos 2x+\cos 3x$ … Эту последовательность можно представить рекуррентным соотношением $b_0=0$, $b_n=b_{n-1}+\cos nx$ *(1)*. Теперь сумму можно представить следующим образом $S_n=\frac{b_1}{1}+\frac{b_2}{2}+\frac{b_3}{3}+\ldots+\frac{b_n}{n}$, а для нее справедливо рекуррентное соотношение $S_0=0$, $S_n=S_{n-1}+\frac{b_n}{n}$ *(2)*. При составлении программы мы будем использовать формулы (1-2)

```
program example6_5;
 var i, n: integer;
     x, b, s: real;
begin
 writeln('Введите показатель степени'); readln(n);
 writeln('Введите основание степени'); readln(x);
 b:=0; s:=0;
for i:=1 to n do
  begin b:=b+cos(i*x); s:=s+b/i; end;
writeln('s= ', s);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Значение b*	*Значение s*
1	i=1	$\cos x$	$\frac{\cos x}{1}$
2	i=2	$\cos x+\cos 2x$	$\frac{\cos x}{1}+\frac{\cos x+\cos 2x}{2}$
3	i=3	$\cos x+\cos 2x+\cos 3x$	$\frac{\cos x}{1}+\frac{\cos x+\cos 2x}{2}+\frac{\cos x+cos2x+\cos 3x}{3}$
…	…		
n	i=n	$\cos x+\cos 2x+\cos 3x$	$\frac{\cos x}{1}+\frac{\cos x+\cos 2x}{2}+\ldots+\frac{\cos x+\ldots+\cos nx}{n}$

5. Написать программу для подсчета суммы $S_n=\sum_{i=1}^{n}\frac{(-1)^{i+1}x^i}{i!}$, где x – вещественное число, n – натуральное число.

Указания по решению задачи. Перейдем от сокращенной формы записи к развернутой, получим $S_n=\frac{x}{1!}-\frac{x^2}{2!}+\frac{x^3}{3!}-\ldots+\frac{(-1)^{n+1}x^n}{n!}$. Каждое слагаемое формируется по формуле $a_n=\frac{(-1)^{n+1}x^n}{n!}$. Если в эту формулу подставить n=0, то получим $a_0=\frac{(-1)^1x^0}{0!}=-1$.

Замечание. Запись n! читается как «n факториал». По определению факториала: 0!=1!=1, n!=1*2*3*…*n. n!=(n-1)!n. Таким образом, факториал выражается рекуррентным соотношением.

Чтобы не вводить несколько рекуррентных соотношений (отдельно для числителя, отдельно для знаменателя), выразим последовательность слагаемое рекур-

рентным соотношением вида $a_n = a_{n-1} q$, где q для нас пока не известно. Найти его можно из выражения $q = \frac{a_n}{a_{n-1}}$. Произведя расчеты мы получим, что $q = -\frac{x}{i}$. Следовательно, для последовательности слагаемых мы получили рекуррентное соотношение $a_0 = -1$, $a_i = -a_{i-1} \cdot \frac{x}{i}$ *(3)*. А всю сумму, по аналогии с предыдущими примерами, можно представить рекуррентным соотношением: $S_0=0$, $S_n = S_{n-1} + a_n$ *(4)*. Таким образом, при составлении программы мы будем пользоваться формулами (3-4).

```
program example6_6;
var  i, n: byte;
    x, a, s: real;
begin
writeln('Введите n'); readln(n);
writeln('Введите x'); readln(x);
s:=0; a:=-1;
for i:=1 to n do
  begin a:=-a*x/i;  s:=s+a; end;
 writeln('s=',s);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Значение a*	*Значение s*
1	i=1	$-(-1)\frac{x}{1} = \frac{x^1}{1!}$	$0 + \frac{x^1}{1!} = \frac{x^1}{1!}$
2	i=2	$-\frac{x}{1} \cdot \frac{x}{2} = -\frac{x^2}{2!}$	$\frac{x^1}{1!} - \frac{x^2}{2!}$
3	i=3	$-(-\frac{x^2}{2!}) \cdot \frac{x}{3} = \frac{x^3}{3!}$	$\frac{x^1}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!}$
…	…	…	…
n	i=n	$-\frac{(-1)^n x^{n-1}}{(n-1)!} \cdot \frac{x^n}{n} = \frac{(-1)^{n+1} x^n}{n!}$	$\frac{x^1}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{(-1)^{n+1} x^n}{n!}$

6. Написать программу для подсчета произведения $P_k = \prod_{n=1}^{k} (1 + \frac{x^{2n} + x^n}{n})$, где x – вещественное число, n – натуральное число.

Указания по решению задачи. Преобразуем заданное выражение к виду $P_k = \prod_{n=1}^{k} (1 + \frac{x^n (x^n + 1)}{n})$ и перейдем от сокращенной формы записи к развернутой

$$P_k = (1 + \frac{x^1(x^1+1)}{1})(1 + \frac{x^2(x^2+1)}{2})(1 + \frac{x^3(x^3+1)}{3})\ldots(1 + \frac{x^k(x^k+1)}{k})$$

В числителе каждой дроби встречается x^n (см.пример 2), его можно вычислить по рекуррентному соотношению $b_0=1$, $b_n = b_{n-1} * x$ *(5)*. Тогда произведение можно представить как $P_k = (1 + \frac{b_1(b_1+1)}{1})(1 + \frac{b_2(b_2+1)}{2})\ldots(1 + \frac{b_k(b_k+1)}{k})$, что в свою очередь

можно выразить рекуррентным соотношением $P_0=1$, $P_k = P_{k-1} * (1 + \frac{b_k(b_k+1)}{k})$ *(6)*.

При составлении программы мы будем пользоваться формулами (5-6).

```
program example6_7;
var  i, n: byte;
    x, b, p: real;
begin
 writeln('Введите n'); readln(n);
 writeln('Введите x'); readln(x);
 p:=1; b:=1;
 for i:=1 to n do
  begin  b:=b*x;   p:=p*( 1+b*(b+1)/i); end;
 writeln('p=',p);
end.
```

Рассмотрим пошаговое выполнение программы:

№ шага	*Значение счетчика*	*Значение b*	*Значение p*
1	i=1	x	$1 * (1 + \frac{x(x+1)}{1})$
2	i=2	x^2	$1 * (1 + \frac{x(x+1)}{1})(1 + \frac{x^2(x^2+1)}{2})$
3	i=3	x^3	$1 * (1 + \frac{x(x+1)}{1})(1 + \frac{x^2(x^2+1)}{2})(1 + \frac{x^3(x^3+1)}{3})$
…	…	…	…
N	i=k	x^k	$1 * (1 + \frac{x(x+1)}{1})(1 + \frac{x^2(x^2+1)}{2})\ldots(1 + \frac{x^k(x^k+1)}{k})$

6.2. Вычисление бесконечных сумм

Будем теперь рассматривать бесконечную сумму вида $u_1(x) + u_2(x) + \ldots + u_n(x) + \ldots = \sum_{i=1}^{\infty} u_i(x)$. Это выражение называется функциональным рядом. При различных значениях x из функционального ряда получаются различные числовые ряды $a_1 + a_2 + \ldots + a_n + \ldots = \sum_{i=1}^{\infty} a_i$. Числовой ряд может быть сходящимся или расходящимся. Совокупность значений x, при которой функциональный ряд сходится, называется его областью сходимости.

Числовой ряд называется сходящимся, если сумма n первых его членов $S_n = a_1 + a_2 + \ldots + a_n$ при $n \to \infty$ имеет предел, в противном случае, ряд называется расходящимся. Ряд может сходиться лишь при условии, что общий член ряда a_n при неограниченном увеличении его номера стремится к нулю: $\lim_{n \to \infty} a_n = 0$. Это необходимый признак сходимости для всякого ряда.

В случае бесконечной суммы будем вычислять ее с заданной точностью e. Считается, что требуемая точность достигается, если вычислена сумма нескольких первых слагаемых и очередное слагаемое оказалось по модулю меньше чем e, то

есть это слагаемое на результат практически не влияет. Тогда его и все последующие слагаемые можно не учитывать.

Написать программу для подсчета суммы $\sum_{i=1}^{\infty}\frac{(-1)^i}{i!}$ с заданной точностью e (e>0).

Указание по решению задачи. Рассмотрим, что представляет из себя заданный ряд: $\sum_{i=1}^{\infty}\frac{(-1)^i}{i!}=-\frac{1}{1}+\frac{1}{2}-\frac{1}{6}+\frac{1}{24}-\frac{1}{120}+\ldots+\frac{1}{\infty}$. Как видим, общий член ряда с увеличением значения i стремится к нулю. Поэтому данную сумму можно вычислить, но только с определенной точностью *e*. Заметим также, что последовательность слагаемых можно выразить с помощью рекуррентного соотношения $a_1=-1$, $a_i=\frac{-a_{i-1}}{i}$, а всю суммы - с помощью рекуррентного соотношения $S_0=0$, $S_n=S_{n-1}+a_n$. (Данные рекуррентные соотношения выведите самостоятельно.)

```
program example6_7;
 var  i: integer;  a, s, e: real;
begin
 writeln('Введите требуемую точность вычисления'); readln(e);
 s:=0; a:= -1; {определяем начальное значение суммы  и значение первого слагаемого }
 i:=1;          {определяем номер вычисленного слагаемого}
 while abs(a)>=e do        {до тех пор, пока очередное слагаемое больше e}
  begin
   s:=s+a;   {добавляем его к сумме}
   i:=i+1;   a:=-a/i;      {вычисляем номер очередного  слагаемого и его значение}
  end;
 writeln('Сумма с заданной точностью равна ', s:6:2 )
end.
```

Рассмотрим подробно, как выполняется цикл при $e = 0.01$:

№ шага	*Значения переменных до выполнения цикла*			*Условие abs(a)>=e*	*Значения переменных после выполнения цикла*		
	i	*a*	*s*		*i*	*a*	*s*
1	1	-1	0	TRUE	2	0.5	-1
2	2	0.5	-1	TRUE	3	-0.166...	-0.5
3	3	-0.166...	-0.5	TRUE	4	0.046...	-0.666...
4	4	0.046...	-0.666...	TRUE	5	0.0083...	-0.62...
5	5	0.0083...	-0.62...	FALSE	5	0.0083...	-0.62...

После выполнения программы на экране будет выведено следующие сообщение «Сумма с заданной точностью равна -0.62».

6.3. Упражнения

I. Объясните, что будет напечатано программой:

1)
```
program e1;
  var  i, n, s: integer;
 begin
  n:=5; s:=0;
  for i:=1 to n do s:=s+(i+1)*i;
  writeln('s=',s);
 end.
```

2)
```
program e2;
  var  i, n:integer;
       b:real;
 begin
  n:=4; b:=1;
  for i:=2 to n do  b:=b*(i-1)/i; writeln('b=', b);
 end.
```


3)
```
program e3;
  var  i, n, a, s: integer;
 begin
  s:=0; a=1; n:=5;
  for i:=1 to n do
   begin
    a:=4*a; s:=s+a;
   end;
  writeln('s=',s);
 end.
```

4)
```
program e4;
  var  i: integer;
       a, s, e: real;
 begin
  s:=0; a:= 1; i:=1; e:=0.01;
  while abs(a)>=e do
   begin s:=s+a; i:=i+2;
         a:=-2*a/sqr(i); end;
  writeln('s=', s:6:4)
 end.
```

II.Объясните и исправьте ошибки в каждом фрагменте программы:

a) $S=1\cdot3+2\cdot4+3\cdot5+\ldots+10\cdot12$

```
s:=0;
for i:=1 to 12 do s:=s+i*(i+1);
```

b) $P=1\cdot3\cdot5\cdot7\cdot\ldots\cdot11$

```
b:=0;
for i:=1 to 11 do b:=b*(2*i-1);
```

c) $S=3^2-3^4+3^6-3^8+3^{10}$

```
s:=0; a=9;
for i:=1 to 5 do
 begin
  a:=a*a; s:=s+a;
 end;
```

d) $S=\sum_{i=0}^{\infty}\frac{1}{2i!}$

```
s:=0; a:= 1; i:=1; e:=0.01;
while abs(a)>=e do
  begin s:=s+a; i:=i+1; a:=a/i; end;
```

III. Для заданного натурального n и действительного x подсчитать следующие суммы:

1. $S=1^2+2^2+3^2+\ldots+n^2$;
2. $S=\sqrt{1}+\sqrt{2}+\sqrt{3}+\ldots+\sqrt{n}$;
3. $S=1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}$;
4. $S=1+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2}$;
5. $S=1+\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{3}}+\ldots+\frac{1}{\sqrt{n}}$;
6. $S=\frac{1}{\sin1}+\frac{1}{\sin2}+\ldots+\frac{1}{\sin n}$;
7. $S=1+2+2^2+2^3+\ldots+2^n$;
8. $S=\cos1-\cos2+\cos3-\ldots+(-1)^{n+1}\cos n$;
9. $S=1!+2!+3!+\ldots+n!$;
10. $S=1-3+3^2-3^3+\ldots(-1)^n3^n$;
11. $S=1!-2!+3!-\ldots+(-1)^{n+1}n!$;
12. $S=\sin x+\sin x^2+\sin x^3+\ldots+\sin x^n$;
13. $S=1+\frac{1}{2!}+\frac{1}{3!}+\ldots+\frac{1}{n!}$;
14. $S=-\frac{1}{2}+\frac{1}{2^2}-\frac{1}{2^3}+\ldots+\frac{(-1)^n}{2^n}$;
15. $S=1^3-2^3+3^3\ldots+(-1)^{n+1}n^3$;
16. $S=x+3x^3+5x^5+7x^7+\ldots+(2n-1)x^{2n+1}$;
17. $S=\frac{\cos x}{1}+\frac{\cos^2 x}{2}+\frac{\cos^3 x}{3}+\ldots+\frac{\cos^n x}{n}$;
18. $S=\frac{1}{3^2}-\frac{1}{5^2}+\frac{1}{7^2}-\ldots+\frac{(-1)^{n+1}}{(2n+1)^2}$
19. $S=\frac{1}{\sin1}+\frac{1}{\sin1+\sin2}+\ldots+\frac{1}{\sin1+\sin2+\ldots+\sin n}$;
20. $S=\sin x+\sin\sin x+\sin\sin\sin x+\ldots+\underbrace{\sin\sin\sin\ldots\sin x}_{n\ раз}$;

IV. Для заданного натурального k и действительного x подсчитать следующие выражения:

1. $S=\sum_{n=1}^{k}\frac{x^n}{n}$;
2. $S=\sum_{n=1}^{k}\frac{2^n\cdot n!}{n^2}$;
3. $S=\sum_{n=1}^{k}\frac{(-1)^{n+1}}{n^2}$;
4. $S=\sum_{n=1}^{k}\frac{x^{2(n-1)}}{(2+4(n-1))^2}$;
5. $S=\sum_{n=1}^{k}\frac{(-1)^{n+1}x^{2n-1}}{(2n-1)!}$;
6. $S=\sum_{n=1}^{k}\frac{1}{n\cdot n!}$;
7. $S=\sum_{n=0}^{k}\frac{(-1)^{n+1}x^{2n+1}}{(2n+1)!}$;
8. $S=\sum_{n=1}^{k}\frac{(-1)^{n-1}x^n}{(2n)!}$;
9. $S=\sum_{n=1}^{k}\frac{(-1)^{n-1}x^{2n}}{(2n)!}$;
10. $S=\sum_{n=1}^{k}\frac{(-1)^n x^n}{2^n 7n}$;
11. $P=\prod_{n=1}^{k}(1+\frac{x^n}{n^2})$;
12. $P=\prod_{n=1}^{k}(1+\frac{x^{2n+1}}{n(n+1)})$;
13. $P=\prod_{n=1}^{k}(1-\frac{x^n}{n!})$
14. $P=\prod_{n=1}^{k}(1+\frac{(-1)^n x^{2n}}{n^3})$;
15. $P=\prod_{n=1}^{k}(1+\frac{(-1)^{n+1}x^n}{n!})$;
16. $P=\prod_{n=1}^{k}(1+\frac{x^{2n}}{n(n+4)})$;
17. $P=\prod_{n=1}^{k}(1+\frac{(-1)^n x^{2n+1}}{n^3+n^2})$;
18. $P=\prod_{n=1}^{k}(1+\frac{x^n}{2n!})$;
19. $P=\prod_{n=2}^{k}(1+\frac{(-1)^n x^{2n-1}}{n^3-1})$;
20. $P=\prod_{n=0}^{k}(1+\frac{(-1)^{n-1}x^{2n}}{(n+2)(n+1)})$;

V. Вычислить бесконечную сумму ряда с заданной точностью ε ($\varepsilon>0$).

1. $\sum_{i=1}^{\infty}\frac{1}{i^2}$
2. $\sum_{i=1}^{\infty}\frac{1}{(i+1)^3}$
3. $\sum_{i=2}^{\infty}\frac{(-1)^i}{i^2-1}$
4. $\sum_{i=1}^{\infty}\frac{1}{i(i+1)}$
5. $\sum_{i=2}^{\infty}\frac{5}{(i+1)(i-1)}$
6. $\sum_{i=1}^{\infty}\frac{(-2)^{i+1}}{i(2i+1)}$
7. $\sum_{i=1}^{\infty}\frac{2}{i!}$
8. $\sum_{i=1}^{\infty}\frac{1}{(2i)!}$
9. $\sum\frac{(-1)^i}{(2i-1)!}$
10. $\sum_{i=1}^{\infty}\frac{(-1)^{i+1}}{2i!}$
11. $\sum_{i=1}^{\infty}\frac{(-1)^{2i}}{i(i+1)(i+2)}$
12. $\sum_{i=3}^{\infty}\frac{(-1)^{2i-1}}{i(i-1)(i-2)}$
13. $\sum_{i=1}^{\infty}\frac{(-3)^{2i}}{3i!}$
14. $\sum_{i=1}^{\infty}\frac{(-5)^{2i-1}}{5(2i-1)!}$
15. $\sum_{i=1}^{\infty}\frac{1}{2^i}$
16. $\sum_{i=1}^{\infty}\frac{1}{3^i+4^i}$
17. $\sum_{i=1}^{\infty}\frac{1}{5^i+4^{i+1}}$
18. $\sum_{i=1}^{\infty}\frac{(-1)^i}{2^{2i}}$
19. $\sum_{i=1}^{\infty}\frac{(-1)^{i-1}}{3^{2i-1}}$
20. $\sum_{i=1}^{\infty}\frac{1}{\sqrt{3^i}}$

6.4. Самостоятельная работа

Задача 1. Для натуральных значений n, m и действительного x вычислить значение выражения $\sum_{i=1}^{n}\prod_{j=1}^{m}\frac{x^i}{j!}$.

Задача 2. Для всех значений $x\in[-1, 1]$ с шагом h вычислить бесконечную сумму $\sum_{n=1}^{\infty}(-1)^{n-1}(1+2^n)\frac{x^n}{n}$. Если для некоторого значения x сумму вычислить нельзя, то вывести об этом сообщение. Результаты представить в виде таблицы:

№	x	S(x)

7. МАССИВЫ

7.1. Описание и обращение к элементам массива

Массив – это фиксированное количество упорядоченных элементов одного и того же типа, объединенных одним именем, где каждый элемент снабжен индексами (одним или несколькими). Массив может быть одномерным и многомерным. В данном пособии будут рассмотрены одномерные и двумерные массивы.

Пример 1. Схематично одномерный массив *a* из 5 целых чисел можно представить следующим образом:

Номера элементов	1	2	3	4	5
Элементы	24	31	-12	3	-7

Двумерный массив называют еще матрицей или таблицей.

Пример 2. Схематично массив *b* из целых чисел, состоящий из 3-х строк и 2-х столбцов, можно представить следующим образом:

	Номер столбца	
Номер строки	*1*	*2*
1	5	-27
2	-35	17
3	21	3

При описании массива определяют имя, тип индексов (или индекса) массива и тип элементов (базовый тип):

```
var <имя массива>: array [<тип индекса>] of <базовый тип>;
```

Например, одномерный массив *a* (пример 1) и двумерный массив *b* (пример 2) можно описать следующим образом:

```
var a: array[1..5] of integer;
    b: array [1..3, 1..2] of integer;
```

Размерность массива ограничена только размером оперативной памяти. Компоненты массива могут быть любого, в том числе и структурированного типа (за исключением файлового). Индексы могут быть любого порядкового типа, кроме типа *longint*.

К любому элементу массива можно обратиться напрямую, указав имя массива, а затем в квадратных скобках номер требуемого элемента (если массив многомерный, то индексы указываются через запятую). Например,

1) *a[3]* – обращение к 3-му элементу описанному выше массива *a* (пример 1), этот элемент равен *-12*.

2) *b[2, 2]* – обращение к элементу, стоящему на пересечении второй строки и второго столбца в массиве *b* (пример 2), этот элемент равен *17*.

3) *b[3, 1]* – обращение к элементу, стоящему на пересечении третьей строки и первого столбца в массиве *b*, этот элемент равен *-21*.

7.2. Ввод-вывод элементов массива

Так как массив хранит не одно значение, а сразу несколько, то ввод и вывод данных из массива производится поэлементно. Для этого обычно используется цикл *for*. Приведем фрагмент программы, в котором вводится, а затем выводится массив *a* из примера 1:

```
for i:=1 to 5 do        {блок ввод элементов массива}
 begin write('a[',i,']='); readln(a[i]); end;

for i:=1 to 5 do        {блок вывода элементов массива}
  write('  a[',i,']=',a[i]);
```

Для ввода и вывода двумерного массива обычно используются вложенные операторы цикла *for*. Приведем фрагмент программы, в котором вводится, а затем выводится двумерный массив *b* из примера 2:

```
for i:=1 to 3 do        {ввод массива}
 for j:=1 to 2 do
 begin write('b[', i, ',', j, '] ='); readln(b[i, j]);  end;

for i:=1 to 3 do        {вывод массива}
 begin
  for j:=1 to 2 do  write(b[i,  j], ' ');
  writeln;
 end;
```

Обратите внимание на то, что параметр внешнего цикла *i* отвечает в массиве за строки, а параметр внутреннего цикла *j* за столбцы. Таким образом, внешний цикл указывает, сколько строк нужно напечатать, а внутренний – сколько элементов в каждой строке. Внутренний цикл выводит числа в строчку, а после того как он завершит работу, управление будет передано оператору *writeln*, который переведет курсор на экране на новую строку. Поэтому данные массива будут выведены на экран в виде таблицы. Подумайте, что будет выведено на экран, если поменять местами внутренний и внешний циклы.

7.3. Примеры использования одномерных массивов

Одномерные массивы удобно использовать тогда, когда данные можно представить в виде некоторой последовательности, элементы которой проиндексированы. В данном разделе мы будем использовать одномерные массивы для хранения

значений некоторой последовательности чисел, нумерация которых начинается с единицы.

Замечания

В общем случае:

1) в массиве могут храниться не только числа, но строки, структуры, а для хранения значений элементов последовательности могут использоваться не только массивы, но файлы, списки;
2) в качестве индекса можно использовать любой порядковый тип данных, и, следовательно, индексацию можно начинать с любого значения входящего в диапазон выбранного порядкового типа.

Все рассмотренные примеры можно будет разделить на три группы:

1. выбор из всего массива элементов по какому-либо признаку (например, нечетных или отрицательных) для какого-либо действия над ними (например, подсчет суммы, количества, вывода номеров, замена таких элементов);
2. поиск наибольшего или наименьшего элементов в массиве;
3. перестановка элементов в массиве.

1. Дана последовательность из 10 целых чисел. Написать программу, которая заменяет в данной последовательности все отрицательные элементы нулями.

```
program example7_1;
 var  a: array[1..10] of  integer;
      i: integer;
begin
 writeln('Введите десять целых чисел');
 for i:=1 to 10 do              {блок ввода данных в массив}
  begin  write('a[',i,']='); readln(a[i]);  end;
 for i:=1 to 10 do              {блок обработки данных в массиве}
  if a[i]<0 then a[i]:=0;  {если элемент массива отрицательный, то заменяем его нулем}
 for i:=1 to 10 do              {блок вывода массива на экран}
   write('  a[',i,']=',a[i]);
end.
```

Результат работы программы:

Исходные данные	*Измененные данные*
2 –4 1 2 –2 0 23 –12 1 -1	2 0 1 2 0 0 23 0 1 0

При решении данной задачи ввод данных и обработка данных были разбиты на два цикла. В данном случае их можно объединить в один, потому что обработка данных не зависит от того введен массив полностью или нет.

Кроме того, данная программа всегда будет обрабатывать только 10 чисел. Иногда требуется обрабатывать переменное число элементов последовательности. В этом случае возникает «проблема» описания массива. Описать его таким образом, чтобы размерность была переменной (например, *a:array [1..n] of integer*, где *n* размерность не была описана как константа) нельзя, т.к. значение переменной *n* не определено, а программа должна четко знать, сколько памяти нужно выделить под массив. Поэтому можно поступить следующим образом: задать размерность массива числом, заведомо большим того, сколько элементов может быть, а потом в программе ввести количество элементов, меньшее, чем это число. Перепишем данную программу, с учетом этих поправок.

```
program example7_2;
 var  a: array[1..100] of  integer;
      i, n: integer;   {n не может быть больше 100}
```

```
begin
 writeln('Введите число элементов последовательности'); readln(n);
 writeln('Ввод элементов последовательности');
 for i:=1 to n do      {ввод и обработка данных}
  begin
    write('a[',i,']='); readln(a[i]);   {ввод очередного элемента}
    if a[i]<0 then a[i]:=0;             {замена отрицательного элемента нулем}
 end;
 for i:=1 to n do   write('  a[',i,']=',a[i]);   {вывод результатов на экран}
end.
```

2. Дана последовательность из n действительных чисел (n≤100). Написать программу для подсчета суммы этих чисел.

```
program example7_3;
 var  a: array[1..100] of  real;
      i, n: integer;
      s:real;
begin
 writeln('Введите число элементов последовательности'); readln(n);
 s:=0;
 for i:=1 to n do
  begin
    write('a[',i,']='); readln(a[i]);   {ввод очередного элемента в массив}
    s:=s+a[i];                          {добавление значения элемента массива к сумме}
 end;
 writeln('s=', s:4:1);
end.
```

Результат работы программы:	*n*	*Исходные данные*	*Ответ*
	5	2.3 0 2.5 1.7 -1.5	S = 5.0

Обратите внимание на то, что при подсчете суммы мы использовали уже известный нам из раздела 6 прием накопления суммы *s:=s+a[i]*.

3. Дана последовательность из n целых чисел (n≤100). Написать программу для подсчета среднего арифметического четных значений данной последовательности.

```
program example7_4;
 var   a: array[1..100] of  integer;
 i, n, k, s: integer;   {k – переменная для подсчета количества четных}
 sr: real;              {элементов; s - для подсчета суммы четных элементов}
begin
 writeln('Введите число элементов последовательности'); readln(n);
 s:=0; k:=0;
 for i:=1 to n do
  begin
    write('a[',i,']='); readln(a[i]); {ввод в массив очередного элемента последовательности}
    if a[i] mod 2 = 0 {если остаток от деления очередного элемента на 2 равен 0,}
     then  begin      { то элемент является четным числом, поэтому  добавить его }
             s:=s+a[i]; k:=k+1; {к сумме и увеличить количество четных  элементов на 1}
           end;
  end;
if k<>0 {если k не нулевое, то четные числа в последовательности есть}
 then begin  { и можно вычислить их среднее арифметическое значение}
        sr:=s/k; writeln('sr=',sr: 5: 2); end
```



```
 else writeln('четных чисел в последовательности нет');
end.
```

Результат работы программы:	*n*	*Исходные данные*	*Ответ*
	5	1 3 7 -41 9	четных чисел в последовательности нет
	4	2 4 6 4	sr = 4.00

4. Дана последовательность из n целых чисел (n≤100). Написать программу, которая определяет наименьшее значение последовательности и его порядковый номер.

```
program example7_5;
 var  a: array[1..100] of  integer;
      i, n, k, min, nmin: integer;
begin
 writeln('Введите число элементов последовательности'); readln(n);
 for i:=1 to n do
  begin write('a[',i,']='); readln(a[i]); end;
  min:=a[1];  {в качестве наименьшего значения полагаем первый элемент массива}
  nmin:=1;    {соответственно его порядковый номер равен 1}
  for i:=2 to n do     {перебираем все элементы массива со второго по последний}
   if a[i]<min  {если очередной элемент окажется меньше значения min,}
    then begin  {то в качестве нового наименьшего значения запоминаем значение }
           min:=a[i];  nmin:=i;     {этого элемента  массива и, соответственно,}
         end;          {запоминаем его номер}
  writeln('наименьший элемент последовательности: ', min, ' его номер: ', nmin);
end.
```

Результат работы программы:	*n*	*Исходные данные*	*Наименьшее значение*	*Его номер*
	5	1 3 7 -41 9	-41	4

5. Дана последовательность из n действительных чисел (n≤100). Написать программу, которая меняет местами в этой последовательности наибольший и наименьший элемент местами (считается, что в последовательности только один наибольший и один наименьший элементы).

```
program example7_6;
var  a: array[1..100] of  real;
     i, n, nmax, nmin: integer;     {nmax, nmin – номера наибольшего и наименьшего}
     max, min:real;                 { элементов соответственно, max, min – их значения}
begin
 writeln('Введите число элементов последовательности'); readln(n);
 writeln('Ввод элементов последовательности');
 for i:=1 to n do
  begin write('a[',i,']='); readln(a[i]); end;
 max:=a[1]; nmax:=1;
 min:=a[1]; nmin:=1;
 for i:=2 to n do      {поиск наибольшего и наименьшего значения в массиве и их номеров}
   begin
    if a[i]<min then begin min:=a[i]; nmin:=i; end;
    if a[i]>max then begin max:=a[i]; nmax:=i; end;
   end;
  a[nmin]:=max;  {в позицию наименьшего элемента записываем значение наибольшего}
  a[nmax]:=min   {в позицию наибольшего элемента записываем значение наименьшего}
  for i:=1 to n do
    write('  a[',i,']=',a[i]:5:3);          {вывод результатов на экран}
end.
```

Результат работы программы:

n	*Исходные данные*	*Измененные данные*
4	1.1 3.4 –41.2 9.9	1.1 3.4 9.9 –41.2

6. Дана последовательность из n действительных чисел (n≤100). Написать программу, которая подсчитывает количество пар соседних элементов последовательности, для которых предыдущий элемент равен последующему.

```
program example7_7;
 var  a: array[1..100] of  real;
      i, n, k: integer;
begin
 writeln('Введите число элементов последовательности');  readln(n);
 for i:=1 to n do
   begin write('a[',i,']='); readln(a[i]); end;
 for i:=1 to n-1 do
   (если соседние элементы равны, то количество искомых пар увеличиваем на 1)
    if a[i]=a[i+1] then k:=k+1;
 writeln('k=',k);
end.
```

Результат работы программы:

n	*Исходные данные*	*Ответ*
6	1.1 3.4 –41.2 9.9 3.1 -3.7	k=0
6	1.1 1.1 –41.2 -3.7 -3.7 -3.7	k=3

Обратите внимание на то, что в последнем цикле параметр i принимает значения от 1 до n-1, а не до n. Это связано с тем, что для i=n-1 существует пара (n-1, n), а для i=n пары (n, n+1) не существует, так как в массиве всего n элементов.

7.4. Примеры использования двумерных массивов

Двумерные массивы находят свое применение тогда, когда исходные данные представлены в виде таблицы, или когда для хранения данных удобно использовать табличное представление.

В данном разделе мы рассмотрим примеры использования двумерных массивов. Как и в случае с одномерными массивами все примеры будут разделены на три группы как и в разделе 7.2.. Поэтому принцип составления программ для решения задач из перечисленных групп такой же, как и в случае одномерного массива. Различия заключаются в следующем:

1) при полном переборе элементов массива используются вложенные циклы *for*, при этом во внешнем цикле обычно перебираются номера строк, во внутреннем – номера столбцов;

2) при обращении к элементу массива в квадратных скобках указывается два индекса: номер строки и столбца.

1. В двумерном массиве, элементами которого являются целые числа, подсчитать среднее арифметическое четных элементов массива.

```
program example7_8;
 var a: array [1..5, 1..5] of integer;
     s, k, n, m, i, j: integer;
     sr: real;
begin
  writeln('Введите размерность массива'); readln(n, m);
```



```
 writeln('Введите элементы массива');
 for i:=1 to n do     (ввод значений в двумерный массив)
  for j:=1 to m do
   begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 s:=0; k:=0;
 for i:=1 to n do
  for j:=1 to m do
    if a[i, j] mod 2 =0 (если элемент массива четный,  то добавляем его к сумме и )
     then  begin      (увеличиваем количество четных элементов на 1)
             s:=s+a[i,  j]; k:=k+1;
           end;
 if k=0 (условие истинно, если ни одного четного элемента в массиве нет)
  then writeln('Четных элементов в массиве нет')
  else begin sr:=s/k; writeln('sr=', sr: 5:2) end;
end.
```

Результат работы программы:

n m	*Массив $A_{n,m}$*	*Ответ*
2 3	2 1 3 1 3 6	4.00
3 2	1 3 5 1 7 9	Четных элементов в массиве нет

2. Дан двумерный массив, элементами которого являются целые числа. Найти значение максимального элемента массива.

```
program example7_9;
 var a: array [1..5, 1..5] of integer;
     max, n, m, i, j: integer;
begin
 writeln('Введите размерность массива'); readln(n, m);
 writeln('Введите элементы массива');
 for i:=1 to n do
 for j:=1 to m do
   begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 max:=a[1, 1];      (в качестве максимального элемента берем значение a[1, 1])
 for i:=1 to n do   (просматриваем все элементы массива )
  for j:=1 to m do
    if a[i, j] > max    (если очередной элемент больше значения максимального)
     then  max:=a[i,  j]; (то в качестве максимального запоминаем этот элемент )
writeln('максимальный элемент =', max);
end.
```

Результат работы программы:

n m	*Массив $A_{n,m}$*	*Ответ*
2 3	2 1 3 1 3 6	6

3. Дана квадратная матрица, элементами которой являются вещественные числа. Подсчитать сумму элементов главной диагонали.

Указания по решению задачи. Для элементов, стоящих на главной диагонали характерно то, что номер строки совпадает с номером столбца. Этот факт будем учитывать при решении задачи.

```
program example7_10;
 var a: array [1..5, 1..5] of real;
```

```
    s: real;
    n, i, j: integer;
begin
 writeln('Введите размерность массива'); readln(n);
 writeln('Введите элементы массива');
 for i:=1 to n do
  for j:=1 to n do  begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 s:=0;
 for i:=1 to n do      {просматриваем все элементы массива}
  for j:=1 to n do
   if i=j        {если для элемента номер строки совпадает с номером столбца,}
    then s:=s+a[i, j];        {то добавляем этот элемент к сумме}
 writeln('Сумма элементов главной диагонали = ', s:8:3)
end.
```

В данной программе перебираются все элементы массива и из них выбираются только те, которые стоят на главной диагонали. Это не рационально, особенно для массивов большой размерности. Так как на главной диагонали стоят элементы *a[1, 1], a[2, 2], ..., a[n, n]*, то можно напрямую обращаться к элементам главной диагонали. Т.е. на *i*-том шаге к элементу главной диагонали мы можем обратиться следующим образом *a[i, i]*. При этом нам потребуется только один цикл *for*. С учетом вышесказанного программа будет выглядеть следующим образом:

```
program example7_11;
 var a: array [1..5, 1..5] of real;
     s: real;
     n, i, j: integer;
begin
 writeln('Введите размерность массива'); readln(n);
 writeln('Введите элементы массива');
 for i:=1 to n do
 for j:=1 to n do  begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 s:=0;
 for i:=1 to n do  s:=s+a[i, i]; {обращаемся к каждому элементу главной диагонали}
 writeln('Сумма элементов главной диагонали = ', s:8:3)
end.
```

Результат работы программы: *n* = 3; *Массив* $A_{n,n}$: 2.4 -1.9 3.1 / 1.1 3.6 -1.2 / -2.1 4.5 -1.7; *Ответ*: Сумма элементов главной диагонали = 4.300

4. Дана прямоугольная матрица, элементами которой являются вещественные числа. Поменять местами ее строки следующим образом: первую строку с последней, вторую с предпоследней и т.д.

```
program example7_12;
 var a: array [1..5, 1..5] of real;
     z: real;
     n, m, i, j: integer;
begin
 writeln('Введите размерность массива'); readln(n, m);
 writeln('Введите элементы массива');
 for i:=1 to n do
  for j:=1 to m do begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
```



```
 for i:=1 to n div 2 do {блок перестановки строк массива}
  for j:=1 to m do {меняются местами все элементы i-той и (n-i+1)-ой строки}
   begin z:=a[i, j]; a[i, j]:= a[n-i+1,j]; a[n-i+1,j]:=a[i, j]; end;
 writeln('Вывод измененного массива');
 for i:=1 to n do
 begin
  for j:=1 to m do write(a[i, j]:8:2);
  writeln;
 end;
end.
```

Замечание. Обратите внимание на то, что в блоке перестановки строк внешний цикл перебирает не все строки, а только половину. Если бы перебирались все строки, то никакой перестановки не происходило, и массив оставался бы неизменным. Подумайте почему.

Результат работы программы: *n m* = 3 4; *Массив* $A_{n,m}$: 2.4 -1.9 3.1 0.0 / 1.1 3.6 -1.2 3.7 / -2.1 4.5 -1.7 4.9; *Ответ*: -2.1 4.5 -1.7 4.9 / 1.1 3.6 -1.2 3.7 / 2.4 -1.9 3.1 0.0

5. Дана прямоугольная матрица, элементами которой являются целые числа. Для каждого столбца подсчитать среднее арифметическое его нечетных элементов и записать полученные данные в новый массив.

Указания по решению задачи. Массив результатов будет одномерный, а его размерность будет совпадать с количеством столбцов исходной матрицы.

```
program example7_13;
 var a: array [1..5, 1..5] of integer;
     b: array [1..5] of integer;
     s: real;
     n, m, k, i, j: integer;
begin
 writeln('Введите размерность массива'); readln(n, m);
 writeln('Введите элементы массива');
 for i:=1 to n do
  for j:=1 to m do begin  write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 for j:=1 to m do {для каждого j-го столбца}
  begin
   s:=0;      k:=0;  {обнуляем сумму и количество нечетных элементов столбца}
   b[j]:=0;    {устанавливаем начальное значение j-того элемента массива b}
  for i:=1 to n do    {перебираем все элементы j-того столбца}
   if a[i, j] mod 2 <> 0 then begin s:=s+a[i, j]; k:=k+1; end;
   if k<>0     {если количество нечетных элементов не равно нулю,}
    then b[j]:=s/k;   {то в b[j] записываем среднее арифметическое значение нечетных }
  end;                {элементов j-того столбца массива a}
 writeln('Вывод результата: ');
 for j:=1 to m do write(b[ j]:8:2);
end.
```

Результат работы программы: *n m* = 2 3; *Массив* $A_{n,m}$: 2 1 3 / 4 3 6; *Ответ*: Вывод результата: 0.00 2.00 3.00

6. Дана матрица A и вектор X соответствующих размерностей. Нечетные строки матрицы заменить элементами вектора X.

```
program example7_14;
 var a: array [1..5, 1..5] of integer
     x: array [1..5] of integer;
     n, m, i, j: integer;
begin
 writeln('Введите размерность массива A'); readln(n, m);
 writeln('Введите элементы массива A');
 for i:=1 to n do
  for j:=1 to m do begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 writeln('Введите элементы вектора X');
 for j:=1 to m do
  begin write('x[', j, ']='); readln(x[ j]); end;
 i:=1;
 while i<=n do                    {для каждой нечетной стоки}
  begin  {i-ую строку массива A заменяем на вектор X поэлементно}
   for j:=1 to m do a[i, j]:=x[j];
   i:=i+2;    {переходим к следующей нечетной строке}
  end;
 writeln('Вывод измененного массива');
 for i:=1 to n do
  begin
   for j:=1 to m do write(a[i,  j]:5);
   writeln;
  end;
end.
```

Результат работы программы:

n m	*Массив* $A_{n,m}$	*Вектор X*	*Ответ*
5 3	-2 1 13	0 4 7	0 4 7
	1 -3 6		1 -3 6
	3 5 1		0 4 7
	3 15 6		3 15 6
	12 4 -3		0 4 7

7. Даны две матрицы A и B соответствующих размерностей, элементами которых являются целые числа. Найти произведение этих матриц.

Указания по решению задачи. Операция произведения двух матриц определена только для матриц соответствующих размерностей $A_{n,m}$ и $B_{m,l}$, то есть количество столбцов первой матрицы должно совпадать с количеством строк второй. В качестве результата операции получается матрица $C_{n,l} = A_{n,m} * B_{m,l}$. Как видим, количество строк матрицы C совпадает с количеством строк матрицы A, а количество столбцов матрицы C равно количеству столбцов матрицы B. Каждый элемент искомой матрицы C определяется по формуле $c[i, j] = \sum_{k=1}^{n} a[i,k] * b[k, j]$, где индекс i принимает значения от 1 до m, а индекс j – от 1 до l.

```
program example7_15;
 var a, b, c: array [1..5, 1..5] of real;
     na, ma, nb, mb, i, k, i, j: integer;
begin
 writeln('Введите размерность массива A'); readln(na, ma);
 writeln('Введите элементы массива A');
 for i:=1 to na do
```



```
  for j:=1 to ma do
   begin write('a[', i, ',', j, ']='); readln(a[i, j]); end;
 writeln('Введите размерность массива B'); readln(nb, mb);
 {проверяем применима операция произведения для заданных матриц}
 if ma<>nb
  then writeln('к данным матрицам операция неприменима, не совпадает размерность')
  else begin
        writeln('Введите элементы массива B');
        for i:=1 to nb do
         for j:=1 to mb do
          begin write('b[', i, ',', j, ']='); readln(b[i, j]); end;
        {подсчет произведения матриц}
        for i:=1 to na do
         for j:=1 to mb do
          begin
           c[i, j]:=0;
            for k:=1 to ma do  c[i,  j]:= c[i, j]+a[i, k]*b[k,j];
          end;
        writeln('Вывод результата');
        for i:=1 to na do
         begin
          for j:=1 to mb do write(c[i,  j]:8:2);
          writeln;
         end;
  end;{конец else}
end.
```

Результат работы программы:

na ma	*Массив* $A_{na,ma}$	*Nb mb*	*Массив* $B_{nb,mb}$	*Ответ* $C_{na,mb}$
3 2	-2 1	2 3	0 4 7	1 -12 -8
	1 -3		1 -3 6	-3 13 -11
	3 5			5 -3 51
3 3	-2 1 1	2 3	0 4 7	к данным матрицам операция неприменима, не совпадает размерность
	1 -3 3		1 -3 6	
	3 5 4			

7.5. Вставка и удаление элементов в массивах

Напомним, что при описании массива мы определяем его максимальную размерность, которая в дальнейшем изменена быть не может. Однако с помощью вспомогательной переменной можно контролировать текущее количество элементов, которое не может быть больше максимального.

Пример2. Рассмотрим фрагмент программы:

```
var a:array [1..10] of integer;
    i: byte;
...
begin
 n:=5;
 for i:=1 to 5 do a[i]:=i*i;
...
end.
```

В этом случае массив можно представить следующим образом:

n=5

a	1	2	3	4	5	6	7	8	9	10
	1	4	9	16	25	0	0	0	0	0

Так как во время описания был определен массив из 10 элементов, а заполнено только первые 5, то оставшиеся элементы будут нулевые.

Что значит *удалить из одномерного массива* элемент с номером 3? Удаление должно привести к физическому «уничтожению» элемента с номером 3 из массива, при этом общее количество элементов должно быть уменьшено. В этом понимании удаления элемента из массива сам массив должен выглядеть следующем образом

a	1	2	4	5	6	7	8	9	10
	1	4	16	25	0	0	0	0	0

недопустимое состояние

Такое удаление в ТР для массивов *невозможно*. Массив – это статическая величина, максимальная размерность которой не может быть изменена. Более того, после определения индексации элементов массива на этапе описания изменить индексацию не возможно.

Однако «удаление» можно смоделировать сдвигом элементов влево и уменьшением значения переменной, которая отвечает за текущее количество элементов в массиве, на единицу:

n=4

a	1	2	3	4	5	6	7	8	9	10
	1	4	16	25	0	0	0	0	0	0

В общем случае, если мы хотим удалить элемент массива с номером k (всего в массиве n элементов), то нам необходимо произвести сдвиг элементов, начиная с k+1-го на одну позицию влево. Т.е. на k-ое место поставить k+1-й элемент, на место k+1 – k+2-й элемент, …, на место n-1 – n-й элемент. После чего значение n уменьшить на 1. В этом случае размерность массива не изменится, изменится лишь текущее количество элементов, и у нас создастся ощущение, что элемент с номером k удален. Рассмотрим данный алгоритм на примере:

```
program exm7_16;
var a:array [1..10] of integer;
    i,k,n: byte;
begin
writeln('Введите размерность массива'); readln(n);
writeln('Введите элементы массива');
for i:=1 to n do read(a[i]);
writeln('Введите номер элемента, который хотите удалить'); readln(k);
for i:=k to n-1 do a[i]:=a[i+1];      (выполнение сдвига)
n:=n-1;                    (уменьшение текущего количества элементов в массиве)
for i:=1 to n do  write(a[i], ' '); (вывод измененного массива на экран)
end.
```

Рассмотрим теперь операцию *удаления в двумерном массиве*. Размерность двумерного массива также зафиксирована на этапе описания массива. Однако при необходимости можно «смоделировать» удаление целой строки в массиве, выполняя сдвиг всех строк, начиная с k-той на единицу вверх. В этом случае размерность массива не изменится, а текущее количество строк (столбцов) будет уменьшено на единицу. В качестве примера удалим из двумерного массива, строку с номером k.

```
program exm7_17;
var a: array [1..10,1..10] of integer;
    i, j, k, n, m: byte;
```



```
begin
writeln('Введите размерность двумерного массива'); readln(n,m);
writeln('Введите элементы массива');
for i:=1 to n do
 for j:=1 to m do read(a[i,j]);
writeln('Введите номер строки, которую хотите удалить');  readln(k);
for i:=k to n-1 do    (сдвигаем все строки, начиная с номера k, на единицу вверх)
 for j:=1 to m do a[i,j]:=a[i+1,j];
n:=n-1;                (изменяем текущее количество строк в массиве)
for i:=1 to n do
  begin for j:=1 to m do  write(a[i,j], ' ');
      writeln; end;
end.
```

Результат работы программы:	n m k	Исходный массив $A_{4\times4}$	Измененный массив $A_{3\times4}$
	4 4 2	1 1 1 1	1 1 1 1
		2 2 2 2	3 3 3 3
		3 3 3 3	4 4 4 4
		4 4 4 4	

Аналогичным образом можно удалить k-тый столбец в двумерном массиве.

Вернемся к массиву, определенному в примере 2. И подумаем теперь, что значит *добавить элемент в одномерный массив* в позицию с номером k? В этом случае все элементы, начиная с k-ого, должны быть сдвинуты вправо на одну позицию. Однако сдвиг нужно начинать с конца, т.е. на первом шаге на n+1-е место поставить n-ый элемент, потом на n-ое место поставить n-1-й элемент, …, наконец, на k+1 место вставить k-й элемент. Таким образом, копия k-го элемента будет на k+1-м месте и на k-е место можно поставить новый элемент. Затем необходимо увеличить текущее количество элементов на 1.

Рассмотрим массив из примера 1 и в качестве k зададим значение равное 3. В этом случае массив будет выглядеть следующим образом:

k=3

a	1	2	3	4	5	6	7	8	9	10
	1	4	9	9	16	25	0	0	0	0

Теперь в позицию с номером 3 можно поместить новое значение. А текущее количество элементов в массиве становится равным 6. Подумайте, почему сдвиг нужно выполнять с конца массива, а не с начала, как мы это делали в случае удаления элемента из массива.

Рассмотрим программную реализацию данного алгоритма:

```
program exm7_18;
var a:array [1..10] of integer;
    x: integer;
    i, k, n: byte;
begin
writeln('Введите размерность массива'); readln(n);
writeln('Введите элементы массива');
for i:=1 to n do read(a[i]);
writeln('Введите номер элемента для вставки'); readln(k);
if n+1<=10  (если не превысится размерность массива, то)
 then begin
      writeln('new='); read(x); (ввести значение нового элемента)
      for i:=n downto k+1 do a[i+1]:=a[i];      (выполнить сдвиг)
      a[k]:=x;                  (в позицию с номером k вставить новый элемент)
```

```
    n:=n+1;                {увеличить текущее количество элементов в массиве}
     for i:=1 to n do  write(a[i], ' '); {вывод измененного массива на экран}
   end
 else writeln('превышение размера массива');
end.
```

Теперь рассмотрим *добавление столбца в двумерный массив*. Для этого все столбцы после столбца с номером k передвигаем на 1 столбец вправо. Затем увеличиваем количество столбцов на 1. После этого копия столбца с номером k будет находиться в столбце с номером k+1. И, следовательно, k-тый столбец можно заполнить новыми значениями.

Рассмотрим программную реализацию алгоритма:

```
program exm7_19;
var a:array [1..10,1..10] of integer;
    i, j, k, n, m: byte;
begin
 writeln('Введите размерность двумерного массива'); readln(n,m);
 writeln('Введите элементы массива');
 for i:=1 to n do
 for j:=1 to m do  read(a[i,j]);
writeln('Введите номер столбца для вставки');  readln(k);
if k+1<=m
 then begin
   for j:=m downto k do {сдвигаем все столбцы, начиная с номера k, на единицу вправо}
    for i:=1 to n do a[i,j+1]:=a[i,j];
   m:=m+1;  {изменяем текущее количество столбцов в массиве}
   for i:=1 to n do    a[i, k]:=0;  {заполняем  k-тый столбец нулями}
   for i:=1 to n do    {выводим измененный массив}
    begin for j:=1 to m do write(a[i,j], ' ');
          writeln; end;
 end
 else writeln('превышение размерности массива');
end.
```

Результат работы программы:	n m k	Исходный массив $A_{4\times4}$	Измененный массив $A_{4\times5}$
	4 4 2	1 1 1 1	1 0 1 1 1
		2 2 2 2	2 0 2 2 2
		3 3 3 3	3 0 3 3 3
		4 4 4 4	4 0 4 4 4

Аналогичным образом можно провести вставку новой строки в двумерный массив.

7.6. Упражнения

I. Объясните, что будет напечатано программой:

```
1) program e1;
   var  i: byte;
        a: array [1..10] of integer;
   begin
    for i:=1 to 10 do a[i]:=10-i+1;
    for i:=1 to 10 do write( a[a[i]],' ');
   end.
```

```
2) program e2;
    var i: byte; b: array [1..4, 1..4] of integer;
   begin
    for i:=1 to 4 do
     for j:=1 to 4 do  b[i, j]:=2*i-3*j;
    write(b[1,2], ' ', b[4,3], ' ', b[3, 4]);
   end.
```

```
3) program e3;
   var  i, t: integer;
        a: array [1..5] of integer;
   begin
    a[1]:=9; a[2]:=4; a[3]:=10; a[4]:=2; a[5]:=1;
    for i:=1 to 4 do
     if a[i]>a[i+1]
      then begin t:=a[i]; a[i]:=a[i+1]; a[i+1]:=t; end
    for i:=1 to 5 do  write( a[i],' ');
   end.
```

```
4)  program e4;
    var  i: byte;
         b: array [1..4, 1..4] of integer;
    begin
     for i:=1 to 4 do
      for j:=1 to 4 do  b[i, j]:=i+j;
     for i:=1 to 4 do
      for j:=1 to i do
       write(b[i, j], ' ');
    end.
```

II.Объясните и исправьте ошибки в каждом фрагменте программы:

1) `var a: array [1...10] of integer;`

2) `var c: array [1..10, 1..3,4] of integer;`

3) `a(3):=8;`

4) `b[1][2]:=5;`

5)
```
for i:=1 to 4 do
  i:=a[i];
```

6)
```
for i:=1 to 4 do
  for i:=1 to 5 do write (b[i, i]);
```

7) замена положительных элементов нулями
```
for i:=1 to 4 do
  if  i>0 then a[i]:=0;
```

8) подсчет суммы четных элементов
```
for i:=1 to 4 do
 for j:=1 to 4 do if odd(b[i,j]) then s:=s+1;
```

III. Дана последовательность целых чисел.

Замечание. Задачи из данного пункта решить двумя способами, используя одномерный массив, а затем двумерный.

1. Заменить все положительные элементы противоположными им числами.
2. Заменить все элементы, меньшие заданного числа, этим числом.
3. Заменить все элементы, попадающие в интервал [a, b], нулем.
4. Заменить все отрицательные элементы, не кратные 3, противоположными им числами.
5. Все элементы, меньшие заданного числа, увеличить в два раза.
6. Подсчитать среднее арифметическое элементов.
7. Подсчитать среднее арифметическое отрицательных элементов.
8. Подсчитать количество нечетных элементов.
9. Подсчитать сумму элементов, попадающих в заданный интервал.
10. Подсчитать сумму элементов, кратных 9.
11. Подсчитать количество элементов, не попадающих в заданный интервал.
12. Подсчитать сумму квадратов четных элементов.
13. Вывести на экран номера всех элементов больших заданного числа.
14. Вывести на экран номера всех нечетных элементов.
15. Вывести на экран номера всех элементов, которые не делятся на 7.
16. Вывести на экран номера всех элементов, не попадающих в заданный интервал.
17. Определить, является ли произведение элементов трехзначным числом.
18. Определить, является ли сумма элементов двухзначным числом.
19. Вывести на экран элементы с четными индексами (для двумерного массива – сумма индексов должны быть четной).
20. Вывести на экран положительные элементы с нечетными индексами (для двумерного массива – первый индекс должен быть нечетным).

IV. Дана последовательность из n действительных чисел.

Замечание. Задачи из данного пункта решить, используя одномерный массив.

1. Подсчитать количество максимальных элементов.
2. Вывести на экран номера всех минимальных элементов.
3. Заменить все максимальные элементы нулями.
4. Заменить все минимальные элементы на противоположные.
5. Поменять местами максимальный элемент и первый.
6. Вывести на экран номера всех элементов, не совпадающих с максимальным.
7. Найти номер первого минимального элемента.
8. Найти номер последнего максимального элемента.
9. Подсчитать сумму элементов, расположенных между максимальным и минимальным элементами (минимальный и максимальный элементы в массиве единственные). Если максимальный элемент встречается позже минимального, то выдать сообщение об этом.
10. Найти номер первого максимального элемента.
11. Найти номер последнего минимального элемента.
12. Подсчитать сумму элементов, расположенных между первым максимальным и последним минимальными элементами. Если максимальный элемент встречается позже минимального, то выдать сообщение об этом.
13. Поменять местами первый минимальный и последний максимальный элементы.
14. Найти максимум из отрицательных элементов.
15. Найти минимум из положительных элементов.
16. Найти максимум из модулей элементов.
17. Найти количество пар соседних элементов, разность между которыми равна заданному числу.
18. Подсчитать количество элементов, значения которых больше значения предыдущего элемента.
19. Найти количество пар соседних элементов, в которых предыдущий элемент кратен последующему.
20. Найти количество пар соседних элементов, в которых предыдущий элемент меньше последующего.

V. Дан массив размером n×n, элементы которого целые числа.

1. Подсчитать среднее арифметическое нечетных элементов, расположенных выше главной диагонали.
2. Подсчитать среднее арифметическое четных элементов, расположенных ниже главной диагонали.
3. Подсчитать сумму элементов, расположенных на побочной диагонали.
4. Подсчитать среднее арифметическое ненулевых элементов, расположенных над побочной диагонали.
5. Подсчитать среднее арифметическое элементов, расположенных под побочной диагональю.
6. Поменять местами столбцы по правилу: первый с последним, второй с предпоследним и т.д.
7. Поменять местами две средних строки, если количество строк четное, и первую со средней строкой, если количество строк нечетное.
8. Поменять местами два средних столбца, если количество столбцов четное, и первый со средним столбцом, если количество столбцов нечетное.

9. Если количество строк в массиве четное, то поменять строки местами по правилу: первую строку со второй, третью – с четвертой и т.д. Если количество строк в массиве нечетное, то оставить массив без изменений.
10. Если количество столбцов в массиве четное, то поменять столбцы местами по правилу: первый столбец со вторым, третий – с четвертым и т.д. Если количество столбцов в массиве нечетное, то оставить массив без изменений.
11. Вычислить A^n, где n – натуральное число.
12. Подсчитать норму матрицы по формуле $\|A\|_1 = \sum_i \max_j a_{i,j}$.
13. Подсчитать норму матрицы по формуле $\|A\|_1 = \sum_j \max_i a_{i,j}$.
14. Вывести элементы матрицы в следующем порядке: 
15. Выяснить, является ли матрица симметричной относительно главной диагонали.
16. Заполнить матрицу числами от 1 до n (где n=m×k, а m – количеcто строк, а k – количество столбцов прямоугольной матрицы) следующим образом: 
17. Определить, есть ли в данном массиве строка, состоящая только из положительных элементов.
18. Определить, есть ли в данном массиве столбец, состоящий только из отрицательных элементов.
19. В каждой строке найти максимум и заменить его на противоположный элемент.
20. В каждом столбце найти минимум и заменить его нулем.

VI. Дан массив размером n×n, элементы которого целые числа.

1. Найти максимальный элемент в каждой строке и записать данные в новый массив.
2. Найти минимальный элемент в каждом столбце и записать данные в новый массив.
3. Четные столбцы таблицы заменить на вектор X.
4. Нечетные строки таблицы заменить на вектор X.
5. Вычислить A*X, где A – двумерная матрица, X – вектор.
6. Для каждой строки подсчитать количество положительных элементов и записать данные в новый массив.
7. Для каждого столбца подсчитать сумму отрицательных элементов и записать данные в новый массив.
8. Для каждого столбца подсчитать сумму четных положительных элементов и записать данные в новый массив.
9. Для каждой строки подсчитать количество элементов, больших заданного числа, и записать данные в новый массив.
10. Для каждого столбца найти первый положительный элемент и записать данные в новый массив.
11. Для каждой строки найти последний четный элемент и записать данные в новый массив.

12. Для каждого столбца найти номер последнего нечетного элемента и записать данные в новый массив.
13. Для каждой строки найти номер первого отрицательного элемента и записать данные в новый массив.
14. Для каждой строки найти сумму элементов с номерами от k1 до k2 и записать данные в новый массив.
15. Для каждого столбца найти произведение элементов с номерами от k1 до k2 и записать данные в новый массив.
16. Для каждой строки подсчитать сумму элементов, не попадающих в заданный интервал, и записать данные в новый массив.
17. Подсчитать сумму элементов каждой строки и записать данные в новый массив. Найти максимальный элемент нового массива.
18. Подсчитать произведение элементов каждого столбца и записать данные в новый массив. Найти минимальный элемент нового массива.
19. Для каждой строки найти номер первой пары неравных элементов. Данные записать в новый массив.
20. Для каждого столбца найти номер первой пары одинаковых элементов. Данные записать в новый массив.

VII. В одномерном массиве, элементы которого – целые числа, произвести следующие действия:

1. Удалить из массива все четные числа.
2. Удалить из массива все максимальные элементы.
3. Удалить из массива все числа, значения которых попадают в данный интервал.
4. Удалить из массива все элементы, последняя цифра которых равна данной.
5. Удалить из массива элементы с номера k1 по номер k2.
6. Вставить новый элемент перед первым отрицательным элементом.
7. Вставить новый элемент после последнего положительного.
8. Вставить новый элемент перед всеми четными элементами.
9. Вставить новый элемент после всех элементов, которые заканчиваются на данную цифру.
10. Вставить новый элемент после всех элементов, кратных своему номеру.
11. Удалить из массива все элементы, в записи которых все цифры различны.
12. Удалить из массива повторяющиеся элементы, оставив только их первые вхождения.
13. Вставить новый элемент после всех максимальных.
14. Вставить новый элемент перед всеми элементами, в записи которых есть данная цифра.
15. Вставить новый элемент между всеми парами элементов, имеющими разные знаки.

VIII. В двумерном массиве, элементы которого – целые числа, произвести следующие действия:

1. Вставить новую строку после строки, в которой находится первый встреченный минимальный элемент.
2. Вставить новый столбец после столбца, в котором нет ни одного отрицательного элемента.

3. Вставить новую строку после всех строк, в которых нет ни одного четного элемента.
4. Вставить новый столбец перед всеми столбцами, в которых встречается заданное число.
5. Вставить строку из нулей после всех строк, в которых нет ни одного нуля.
6. Удалить все строки, в которых нет ни одного четного элемента.
7. Удалить все столбцы, в которых первый элемент больше последнего.
8. Удалить все строки, в которых сумма элементов не превышает заданного числа.
9. Удалить все столбцы, в которых четное количество нечетных элементов.
10. Удалить все строки, в которых каждый элемент попадает в заданный интервал.
11. Удалить все столбцы, в которых все элементы положительны.
12. Удалить все строки, в которых среднее арифметическое элементов является двузначным числом.
13. Удалить все строки и столбцы, на пересечении которых стоят минимальные элементы.
14. Удалить из массива k-тую строку и j-тый столбец, если их значения совпадают.
15. Уплотнить массив, удалив из него все нулевые строки и столбцы.

7.7. Самостоятельная работа

Задача 1. Вычислить n! (n>13).

Указания по решению задачи. Сложность состоит в том, что 8!=40320 (превышение диапазона значений целого типа *integer*), а 13!=6227020800 (превышение диапазона значений целого типа *longint*). Поэтому для вычисления факториала рекомендуем использовать одномерный массив вида:

Номера элементов	0	1	2	3	4	5	6	7	8	9	10
Элементы	10	0	0	8	0	2	0	7	2	2	6

где *a[0]* используются для хранения количества цифр в числе, а *a[i]* для хранения i цифры числа.

Задача 2. Изменить решение *задачи 1* так, чтобы вычислялась степень числа a^n, например 2^{50}.

Задача 3. Дан массив размером n×n. Вывести его элементы при обходе по спирали, начиная с центра.

Задача 4. Латинским квадратом порядка n называется массив размером n×n, каждая строка и каждый столбец которого содержат все числа от 1 до n. Для заданного значения n:

1) проверить, является ли заданный массив латинским квадратом;
2) сформировать латинский квадрат порядка n.

Примеры латинских квадратов:

2	3	4	1
3	4	1	2
4	1	2	3
1	2	3	4

n=4

4	5	1	2	3
2	3	4	5	1
5	1	2	3	4
3	4	5	1	2
1	2	3	4	5

n=5

8. СТРОКИ

8.1. Описание переменных строкового типа

Строкой называется последовательность символов определенной длины. Переменные этого типа определяются путем указания имени переменной и, через двоеточие, зарезервированного слова string. Дополнительно в квадратных скобках можно указать максимальный размер (максимальную длину) строки. Если максимальный размер не указан, то он автоматически принимает значение 255, т.е. максимально возможная строка может состоять из 255 символов. Например, описать строковые переменные можно следующим образом:

```
var s: string[30];
    s1: string;
```

В данном случае строка s может использоваться для хранения не более чем 30 символов, строка s1 - 255 символов.

В памяти под переменную строкового типа будет отводиться n+1 байт, где n – это размер строки. Дополнительный байт используется для служебного назначения. В него записывается текущее количество символов строки. Это значение будет автоматически изменяться во время работы программы засчет добавления или удаления символов из строки. Обратиться к этому байту можно с помощью специальной функции length, о которой мы поговорим чуть позже.

Со строкой можно работать как с одномерным массивом символов или как с самостоятельным объектом через стандартные процедуры и функции. Однако независимо от способа работы вводятся и выводятся строковые переменные не поэлементно, как массивы, а целиком. При этом для ввода строки можно использовать только команду readln. В противном случае будет введена пустая строка. Кроме того, если ввести символов больше чем n –количество символов, указанное при описании данного строкового типа в квадратных скобках, то в строковую переменную будут записаны только первые n символов. Остальные символы будут проигнорированы. Например, в результате выполнения следующей программы:

```
program ex8_1;
 var s: string[3];
begin
 readln(s); writeln(s);
end.
```

при вводе с клавиатуры слова *Маша*, на экран будет выведено слов *Маш*.

8.2. Работа со строками как с одномерными массивами

Так же как и в массиве, к каждому элементу строки можно обратиться по его номеру: s[3], s[5], s[i] и так далее. Такое обращение можно использовать для просмотра или изменения символа строки по указанному номеру.

Дополнительные операции, которые не определены для массивов, но которые очень важны для строк, перечислены в следующей таблице:

Название операции	Знак операции	Результат операции	Пояснения
Объединение	+	строка, состоящая из последовательного присоединения исходных строк	Если s='y', s1='ea', s3='r', то s+s1+s2+s3='year'
Сравнение	<, >, =, ≥, ≤, <>	истина или ложь	Сравнение производится слева направо посимвольно, сравниваются коды соответствующих символов до тех пор, пока не нарушено равенство. Две строки называются равными, если они равны по длине и совпадают посимвольно. Результат следующих сравнений – истина: 'Bal'<'bal' (ord('B')<ord('b')) 'balk'>'bal' (длина первой строки больше)

Рассмотрим приемы работы со строками как с одномерными массивами на примере следующих задач.

1. Подсчитать, сколько раз данный символ встречается в данной строке.

```
program example8_2;
 var  s:string;  c:char; k ,i: integer;
begin
 writeln('Введите строку'); readln(s);
 writeln('Введите символ'); readln(c);
 k:=0;
 for i:=1 to length(s) do     {просматриваем строку посимвольно: если i-тый символ }
  if s[i]=c  then k:=k+1;     { строки s равен символу c, то увеличиваем k на единицу }
writeln('k=',k);
end.
```

Результат работы программы:

s	c	k
информатика	и	2
информатика	у	0

2. В данной строке заменить все вхождения одного символа на другой.

```
program example8_3;
var s:string;  c1, c2:char; i:integer;
begin
 writeln('Введите строку'); readln(s);
 writeln('Введите символ, который будем заменять'); readln(c1);
 writeln('Введите символ, на который будем заменять'); readln(c2);
  for i:=1 to length(s) do   {просматриваем строку посимвольно: если i-тый символ }
   if s[i]=c1 then s[i]:=c2;  { строки s  равен символу c1, то заменяем его на символ c2}
  writeln('s=',s);
end.
```

Результат работы программы:

Исходная строка s	c1	c2	Измененная строка s
информатика	у	ч	информатика
прут	т	д	прут

3. Заменить каждую точку в заданной строке на многоточие.

Указание к решению задачи. Т.к. в каждой позиции строки может храниться только один символ, а нам нужно будет заменять один символ на три символа, то нам потребуется вспомогательная строка.

```
program example8_4;
  var s, s1:string; i :integer;
begin
  writeln('Введите строку'); readln(s);
  s1:='';      {во вспомогательную переменную записываем пустую строку}
  for i:=1 to length(s) do {просматриваем строку посимвольно}
   if s[i]='.' {если символ строки s с номером i равен точке, то}
    then s1:=s1+'...' {к вспомогательной строке приписываем справа многоточие}
    else s1:=s1+s[i]; {иначе приписываем i символ исходной строки}
  writeln('s1=',s1);
end.
```

Результат работы программы:

Исходная строка s	*Измененная строка s*
Вечерело.	Вечерело...
Холодно. Сыро...	Холодно... Сыро.....

4. Удалить из заданной строки все цифры.

Указание к решению задачи. Решение данной задачи также будет проводиться с помощью вспомогательной строки.

```
program ex8_5;
 var s, s1: string; i:byte;
begin
  writeln('Введите строку'); readln(s);
  s1:=''; {во вспомогательную переменную записываем пустую строку}
  for i:=1 to length(s) do
   if not ((s[i]>='0') and (s[i]<='9')) {если i символ строки не является цифрой, то}
    then s1:=s1+s[i];                   {записываем его во вспомогательную строку}
  writeln(s1);
end.
```

Результат работы программы:

Исходная строка s	*Измененная строка s*
Вечерело...	Вечерело...
Встреча назначена на 14.00.	Встреча назначена на ...

5. Выяснить, есть ли в данной строке данная буква. Результатом программы должно быть сообщение «Да» или «Нет».

```
program example8_6;
var s:string;  c:char; i:integer;
    b:boolean;      {переменная, которая будет иметь значение истина, если
begin               {в данной строке есть данная буква, и ложь в противном случае}
 writeln('Введите строку'); readln(s);
 writeln('Введите символ'); readln(c);
 b:=false; i:=1;
 while (i<= length(s)) and not b do      {цикл работает до тех пор, пока не встретится
 begin                                   {данный символ или не закончилась  строка}
  if s[i]=c then b:=true;   {перебираем по очереди символы и если найдется}
  i:=i+1; end;              { искомый, то значением переменной b становится истина}
 if b then writeln('Да') else writeln('Нет');
end.
```

Результат работы программы:

Исходная строка s	*c*	*ответ*
информатика	у	нет
пруд	д	да

6. Дана строка, которая представляет собой слова, разделяющиеся друг с другом одним пробелом. Вывести на экран все слова начинающиеся и заканчивающиеся одной и той же буквой.

```
program example8_7;
 var sl:array[1..30] of string;  {определили массив строк}
     s:string;  i,k:integer;     {k – счетчик слов в строке и  в массиве sl}
begin
 writeln('Введите строку'); readln(s);
 s:=s+' '; {добавляем к строке справа один пробел}
 k:=1; {начинаем формировать первое слово}
 for i:=1 to length(s) do
  if s[i]=' '    {i-тый символ строки пробел, то k-тое  слово закончилось}
   then k:=k+1                {переходим к формированию нового слова}
   else  sl[k]:=sl[k]+s[i];   {иначе добавляем i-тый символ к k-тому слову}
 for i:=1 to k-1 do           {перебираем все слова}
  if sl[i][1]=sl[i][length(sl[i])] {если первый и последний символы i-того символа равны, то}
   then write(sl[i], ' ');         {выводим данное слово на экран}
end.
```

Результат работы программы:

s	*ответ*
Маша пришла в гости в 20.00	
шалаш стоял около воды	шалаш около

8.3. Работа со строками через стандартные процедуры и функции

Наиболее важные стандартные процедуры и функции для работы со строкам перечислены в следующей таблице:

Вызов	Тип результата	Назначение
Length(s)	integer	Функция: возвращает длину строки *s*.
Concat(s1,s2,...,sn)	string	Функция: возвращает строку, представляющую собой объединение строк *s1, s2, s3* и т.д.
Copy(s,i,c)	string	Функция: возвращает подстроку строки *s*, начиная с позиции *i* и состоящую из *c* символов.
Delete(s,i,c)	string	Процедура: удаляет из строки *s c* символов начиная с позиции *i*.
Insert(s1,s,i)	string	Процедура: вставляет подстроку *s1* в строку *s* начиная с позиции *i*.
Pos(s1,s)	integer	Функция: возвращает позицию первого вхождения подстроки *s1* в строку *s* или 0, если *s* не содержит подстроки *s1*.
Str (i, s)	string	Процедура: преобразует число *i* в строку. Результат записывает в переменную *s*.
Val (s, i, code)		Процедура: преобразует строку *s* к числу, тип которого соответствует переменной *i*. Результат записывается в переменную *i*. Если преобразование прошло успешно в переменную *code* записывается значение 0, иначе записывается номер символа, для которого возникла ошибка при преобразовании типов.

Рассмотрим использование данных процедур и функций на примерах. Пусть s1='в саду растут деревья'; s2=' и кусты'. Тогда:

1) Length(s1)=21;
2) Concate(s1,s2)='в саду растут деревья и кусты';

3) Copy(s1,8,6)='растут';
4) Результатом delete(s1,3,4) будет s1='в растут деревья';
5) Результатом insert('парке',s1,3) будет s1='в парке растут деревья';
6) Pos('a',s1)=4; pos('ю',s1)=0;
7) Результатом Str(123, s) будет строка s='123';
8) Результатом Val('12.3', i, code) будет число i=12.3, при этом в переменную code будет записано значение 0;
9) При выполнении процедуры Val('12,3', i, code) в переменную i будет записано значение 0, т.к. преобразовать стоку '12,3' к числовому формату невозможно, а в переменную code будет записано значение 3.

Рассмотрим приемы работы со строками через использование стандартных процедур и функций на примере задач из раздела 8.2 и некоторых новых задач.

1. Подсчитать, сколько раз данный символ встречается в данной строке.

```
program exm8_8;
 var s:string; c:char;
     k,i:byte;
begin
 writeln('введите строку'); readln(s);
 writeln('введите символ'); readln(c);
 k:=0;
 i:=pos(c,s); {определяем позицию символа c в строке s }
 while i<>0 do {пока символ c встречается в строке s}
  begin  k:=k+1; {увеличиваем k}
         delete(s,i,1); {удаляем один символ из стоки s в позиции i}
         i:=pos(c,s); end;{ еще раз ищем символ c в строке s}
writeln(k);
end.
```

2. В данной строке заменить все вхождения одного символа на другой.

```
program example8_9;
 var s: string;
     c1, c2: char;
     i:byte;
begin
 writeln('Введите строку'); readln(s);
 writeln('Введите искомый символ'); readln(c1);
 writeln('Введите символ для замены'); readln(c2);
 if c1<>c2=0 {если символы не равны, то }
 then begin
         i:=pos(c1,s); {определяем позицию первого  символа c1 в строке s}
         while i<>0 do {до тех пор, пока символ c1 встречается в строке s}
          begin delete(s,i, 1);  insert(s, i, c2); {заменяем его на символ c2}
                i:=pos(c1,s); end;{ищем повторное вхождение символа c1 в строке s}
      writeln(s); {выводим преобразованную строку}
     end
  else writeln('ошибка');
end.
```

Подумайте над следующими вопросами:

1) почему в программу добавлена проверка на несовпадение значений c1 и c2?

2) что нужно исправить в программе, чтобы ее можно было применять для замены одной подстроки на другую?

3. Удалить из заданной строки все цифры.

```
program example8_10;
 var s, didgit: string;
     i:byte;
begin
 writeln('Введите строку')readln(s);
 didgit:='0123456789';
 i:=1;
 while i<=length(s) do {пока не просмотрели все символы строки}
  if pos(s[i],didgit)<>0 {проверяем, является ли  i-тый символ строки в цифрой}
   then delete(s,i,1) {если да, то удаляем его}
   else i:=i+1; {если нет, то переходим к проверке следующего символа}
 writeln(s);
end.
```

Подумайте, почему для решения задачи используется цикл *while*, можно ли вместо него использовать цикл *for*.

4. Дана строка, которая представляет собой слова, разделенные одним пробелом. Вывести на экран все слова начинающиеся и заканчивающиеся одной и той же буквой.

```
program example8_11;
 var sl: array[1..20] of string; {определили массив строк}
     s:string;
     i,k:byte; {k – счетчик слов в строке и, соответственно, в массиве sl}
begin
 writeln('Введите строку'); readln(s);
 s:=s+' '; {добавляем к строке справа один пробел}
 i:=pos(' ', s); {определяем индекс первого пробела в строке}
 k:=0;
 while i<>0 do {до тех пор пока в строке встречаются пробелы}
 begin
  k:=k+1; {формируем новое слово}
  sl[k]:=copy(s,1,i-1); {копируем все  символы до пробела , т.е. k-тое слово}
  delete(s,1,i); {удаляем это слово из строки вместе с пробелом}
  i:=pos(' ', s); {ищем новый пробел в строке}
 end;
 for i:=1 to k do       {выводим на экран все слова из массива sl, начинающиеся и }
  if sl[i][1]=sl[i][length(sl[i])] then  writeln(sl[i]);   {оканчивающиеся на одну и ту же букву}
end.
```

5. Дана строка, которая представляет собой слова и целые числа, разделенные одним пробелом. Вывести на экран только целые числа.

```
program example8_12;
 var sl: array[1..20] of string; {определили массив строк}
    s:string;
    i,k:byte;
    a, code:integer;
begin
 writeln('Введите строку'); readln(s);
```

```
 s:=s+' '; {добавляем к строке справа один пробел}
 i:=pos(' ', s); {определяем индекс первого пробела в строке}
 k:=0;
 while i<>0 do {до тех пор пока в строке встречаются пробелы}
 begin
  k:=k+1; {формируем новое слово}
  s[k]:=copy(s,1,i-1); {копируем все символы до пробела , т.е. k-тое слово}
  delete(s,1,i); {удаляем это слово из строки вместе с пробелом}
  i:=pos(' ', s); {ищем новый пробел в строке}
 end;
 for i:=1 to k do {перебираем все элементы массива s}
 begin
  {выполняем преобразование каждого элемента в целочисленное представление,
   результат преобразования записывается в переменную a }
  val(s[i], a, code);
  if code=0 {если преобразование прошло успешно, то code равен 0 }
  then write(a, ' '); {выводим значение a на экран}
 end
end.
```

Результат работы программы:	*Исходная строка*	*Ответ*
	Встреча назначена на 14.00 5 мая 2008 года	5 2008

8.4. Упражнения

I. Объясните, что будет напечатано программой:

1)
```
program ex1;
  var s: string[10];
 begin
  s:='опять в школу…';
  writeln(s);
 end.
```

2)
```
program ex2;
  var s: string[10];
 begin
  s:='опять в школу…';
  writeln(pos('o',s)+pos('o', s));
 end.
```

3)
```
program ex3;
  var s, a: string[10];
      i: byte;
 begin
  s:='мираж'; a:='';
  for i:=1 to length(s) do a:=s[i]+a;
  writeln(a);
 end.
```

4)
```
program ex4;
  var s: string;
      i: byte;
 begin
  s:='казнить нельзя помиловать';
  i:=pos(' ', s);
  insert(s, i, ',');
 end.
```

5)
```
program ex5;
  var s, a: string;
      i: byte;
 begin
  s:='123'; a:='';
  for i:=1 to length(s) do
    a:=s[i]+a+s[i];
  writeln(a);
end.
```

6)
```
program ex6;
  var s, a: string;
      i: byte;
 begin
  s:='123'; a:='';
  for i:= length(s) downto 1 do
   a:=a+s[i]+a;
  writeln(a);
 end.
```


7)
```
program ex7;
  var s: string;
 begin
  s:='кол около колокола';
  i:=pos('оло');
  while i<>0 do
   begin
    delete(s, i, 3); i:=pos('оло');
   end;
  writeln(s);
 end.
```

8)
```
program ex8;
  var s: string; i, k: byte; f: boolean;
 begin
  s:='казнить нельзя помиловать';
  f:=true; i:=length(s);
  while (i>0) and f do
    if s[i]=' '
      then begin k:=i; f:=false; end
      else i:=i-1;
  insert(s, k, ' '); writeln(s);
 end.
```

II. Найти и объяснить ошибки в каждом фрагменте программы:

1)
```
program ex1;
  var s: string;
      i: integer;
 begin
  s:='опять в школу…';
  for i:=0 to length(s) do writeln(s[i]);
 end.
```

2)
```
program ex2;
  var s: char;
      a: string;
 begin
  s:='опять в школу…'; c:='я';
  writeln(pos(c, s));
 end.
```

3)
```
program ex3;
  var s: string;
      i, a: byte;
 begin
  s:='123456789'; a:=0;
  for i:=1 to length(s) do a:=a+s[i];
  writeln(a);
 end.
```

4)
```
program ex4;
  var s: string;
      i: byte;
 begin
  s:= 'кол около колокола';
  for i:=1 to length (s) do
    if s[i]='o' then s[i]:='a' else i:=i+1;
 end.
```

5) *удаление буквы o из строки s*
```
program ex5;
  var s: string;
      i: byte;
 begin
  s:='кол около колокола'; a:=''; i:=1;
  while i<=length(s) do
   if s[i]<>'o'
    then a:=a+s[i] else i:=i+1;
   writeln(a);
 end.
```

5) *дублирование буквы o в строке s*
```
program ex6;
  var s: string; i, k: byte;
      f: boolean;
 begin
  s:='кол около колокола'; a:=''; i:=1;
  while i<=length(s) do
   if s[i]='o'
    then a:=a+s[i]+s[i]  else i:=i+1;
   writeln(a);
 end.
```

III. Разработать программу, которая для заданной строки s:

1. вставляет символ x после каждого вхождения символа y;
2. меняет местами первую букву со второй, третью с четвертой и т.д.
3. определяет, какой из двух символ встречается чаще в данной строке;
4. подсчитывает общее число вхождений символов x и y;
5. подсчитывает количество гласных букв;

6. определяет, имеются ли в строке два соседствующих одинаковых символа.
7. удаляет среднюю букву, если длина строки нечетная, и две средних, если длина строки четная;
8. удваивает каждое вхождение заданного символа x;
9. удаляет все символы x;
10. удаляет все подстроки substr;
11. заменяет все вхождения подстроки substr1 на подстроку substr2;
12. подсчитывает сумму всех содержащихся в ней цифр;
13. подсчитывает количество содержащихся в ней цифр;
14. находит порядковые номера первого и последнего вхождения символа x;
15. заменяет все группы стоящих рядом точек на многоточие;
16. выводит на экран последовательность символов, расположенных до первого двоеточия;
17. выводит на экран последовательность символов, расположенных после последнего двоеточия;
18. удаляет из нее последовательность символов, расположенных между круглыми скобками (считается, что в строке ровно одна пара круглых скобок).
19. удаляет из нее последовательность символов, расположенных между двумя запятыми (считается, что в строке ровно две запятые);
20. определяет, сколько различных символов встречается в строке.

IV. Дана строка, в которой содержится осмысленное текстовое сообщение. Слова сообщения разделяются пробелами и знаками препинания.

1. Вывести только те слова сообщения, в которых содержится заданная подстрока.
2. Вывести только те слова сообщения, которые содержат не более чем n букв.
3. Удалить из сообщения все однобуквенные слова (вместе с лишними пробелами).
4. Удалить из сообщения все повторяющиеся слова.
5. Подсчитать сколько раз заданное слово встречается в сообщении.
6. Найти самое длинное слово сообщения.
7. Найти самое короткое слово сообщения.
8. Вывести на экран все слова-палиндромы, содержащиеся в сообщении.
9. Подсчитать количество слов, содержащихся в сообщении.
10. По правилу расстановки знаков препинания перед каждым знаком препинания пробел отсутствует, а после него обязательно стоит пробел. Учитывая данное правило, проверьте текст на правильность расстановки знаков препинания и, если необходимо, внесите в текст изменения.
11. Вывести только те слова, которые встречаются в тексте ровно один раз.
12. Вывести только те слова, которые встречаются более n раз.
13. Вывести слова сообщения в алфавитном порядке.
14. Вывести слова сообщения в порядке возрастания их длин.
15. Найти максимальное целое число, встречающееся в сообщении.
16. Найти сумму всех имеющихся в тексте чисел (целых и вещественных, причем вещественное число может быть записано в экспоненциальной форме).
17. В сообщении могут встречаться номера телефонов, записанные в формате xx-xx-xx, xxx-xxx или xxx-xx-xx. Вывести все номера телефонов содержащихся в сообщении.

18. В сообщении может содержаться дата в формате дд.мм.гг. В заданном формате дд – целое число из диапазона от 1 до 31, мм – целое число из диапазона от 1 до 12, а гг – целое число из диапазона от 1 до 2020 (если какая-то часть формата нарушена, то данная подстрока в качестве даты не рассматривается). Заменить каждую дату сообщения на дату следующего дня.
19. В сообщении может содержаться время в формате чч:мм:сс. В заданном формате чч – целое число из диапазона от 00 до 24, мм и сс – целые числа из диапазона от 00 до 60 (если какая-то часть формата нарушена, то данная подстрока в качестве даты не рассматривается). Преобразовать каждое время к формату чч:мм, применив правило округления до целого числа минут.
20. В сообщении могут содержаться IP-адреса компьютеров в формате d.d.d.d, где d – целое число из диапазона от 0 до 255. Вывести все IP-адреса содержащиеся в тексте.

8.5. Самостоятельная работа

Задача 1. Известны фамилия, имя и отчество пользователя. Найти его код личности. Правило получения кода личности: каждой букве ставится в соответствие число – порядковый номер буквы в алфавите. Эти числа складываются. Если полученная сумма не является однозначным числом, то цифры числа снова складываются и так до тех пор, пока не будет получено однозначное число. Например:
Исходные данные: Александр Сергеевич Пушкин
Код личности: (1+13+6+12+19+1+15+5+18)+(19+6+18+4+6+6+3+10+25)+(17+21+26+12+10+15)=288 ⇒ 2+8+8=18 ⇒ 1+8=9

Задача 2. В шифре Цезаря алфавит размещается на круге по часовой стрелке. За последней буквой алфавита идет первая буква алфавита, т.е. после буквы «я» идет буква «а». При шифровании текста буквы заменяются другими буквами, отстоящими по кругу на заданное количество позиций (сдвиг) дальше по часовой стрелке. Например, если сдвиг равен 3, то буква «а» заменяется на букву «г», буква «б» на букву «д», а буква «я» на букву «в».

Зашифровать сообщение, используя шифр Цезаря со сдвигом k.

Задача 3. Сообщение, зашифрованное шифром Цезаря достаточно легко расшифровать, зная сдвиг. Однако шифр Цезаря можно расшифровать даже при неизвестном значении сдвига, пользуясь следующим алгоритмом:

1) найти частоты букв в сообщении f_i, где i=1, … n (n-число букв в алфавите);
2) найти вероятности букв в сообщении p_i, где i=1, … n;
3) вычислить $sum(k)=\sum_{i=1}^{n}|p_i - f_i(k)|$, где k – это значение сдвига. Найти минимальное значение sum, подсчитывая ее для различных значений k от 1 до n. Значение k, на котором достигается минимум значения sum, считается сдвигом в шифре Цезаря.
4) использовать найденное значение k для расшифровки заданного сообщения.

9. РЕШЕНИЕ ПРАКТИЧЕСКИХ ЗАДАЧ

9.1. Алгоритмы поиска делителей натурального числа

По определению целое число i является делителем натурального числа N, если при делении N на i остаток от деления равен нулю. Поэтому, чтобы найти все делители числа N, нужно перебрать все натуральные числа от 1 до N и проверить, являются ли они его делителями. Данный алгоритм можно реализовать с помощью следующей программы:

```
program example9_1;
 var n, i: integer;
begin
 writeln('Введите натуральное число n'); readln(n);
 for i:=1 to n do
  if n mod i = 0       {проверка: остаток при делении n на i равен 0?}
   then write(i,' ');  {если да, то число i делитель n}
end.
```

Рассмотрим подробнее, как работает алгоритм на каждом шаге при n = 100:

Значение i	Проверка условия n mod i = 0	Результат (вид экрана)
1	TRUE	1
2	TRUE	1 2
3	FALSE	1 2
4	TRUE	1 2 4
...	...	...
100	TRUE	1 2 4 5 10 20 25 50 100

Рассмотренный алгоритм решает поставленную задачу, выполняя 100 итераций (повторений) цикла при N=100. Данный алгоритм работает неэффективно, т.к. для нахождения делителей перебираются все числа от 1 до N. Однако для любого числа 1 и само число являются его тривиальными делителями. Поэтому из диапазона следует исключить числа 1, N. Более того, наибольшим делителем, отличным от самого числа N, может быть N/2, а все числа большие N/2 заведомо не могут быть его делителями. Поэтому из рассматриваемого диапазона нужно исключить все числа большие N/2. В результате для поиска делителей числа N можно перебирать все натуральные числа из диапазона от 2 до N div 2. В результате диапазон исследуемых чисел сократился в два раза, что для больших значений N дает выигрыш во времени выполнения программы. Усовершенствованный алгоритм можно реализовать следующей программой:

```
program example9_2;
 var n, i: integer;
begin
 writeln('Введите целое число n'); readln(n);
 write(1,' ');
 for i:=2 to n div 2 do
  if n mod i = 0       {проверка: остаток при делении n на i равен 0?}
   then write(i,' ');  {если да, то число i делитель n}
 writeln(n,' ');
end.
```

Рассмотрим подробнее, как работает алгоритм на каждом шаге при n = 100:

	Проверка условия n mod i = 0	Результат (вид экрана)
до выполнения цикла	-	1
i = 2	TRUE	1 2
i = 3	FALSE	1 2
i = 4	TRUE	1 2 4
...		
i = 50	TRUE	1 2 4 5 10 20 25 50
после выполнения цикла	-	1 2 4 5 10 20 25 50 100

Рассмотренный алгоритм решает поставленную задачу, выполняя 50 итераций цикла при N=100. Однако и данный алгоритм можно усовершенствовать, если вспомнить то факт, что если i является делителем числа N, то и число N/i также будет являться его делителем. Например, если число 100 делится на 2, то оно делится и на 50 (т.е. 100/2). Таким образом, почти все делители образуют пару и если мы нашли один делитель, то можем определить и парный ему. Исключение составляют только такие делители, квадраты которых равны самому числу. Например, число 10 является делителем числа 100, но т.к. $10^2=100$, то у этого делителя числа 100 нет парного. Для таких делителей выполняется свойство $i=\sqrt{N}$, поэтому для поиска делителей числа N можно перебирать все натуральные числа из диапазона от 1 до $\sqrt{N}$. Усовершенствованный алгоритм можно реализовать следующей программой:

```
program example9_3;
 var n, i: integer;
begin
 writeln('Введите целое число n'); readln(n);
 for i:=1 to trunc(sqrt(N)) do
  if n mod i = 0              {проверка: остаток при делении n на i равен 0?}
   then if sqr(i)=N           {если у i нет парного делителя, то }
        then write(i,' ')     {выводим только значение i}
        else write(i,' ', N div i, ' '); {иначе выводим i и парный ему делитель}
end.
```

Рассмотрим подробнее, как работает алгоритм на каждом шаге при n = 100:

	Проверка условия N mod i = 0	Проверка условия Sqr(i) = N	Результат (вид экрана)
i = 1	TRUE	FALSE	1 100
i = 2	TRUE	FALSE	1 100 2 50
i = 3	FALSE	-	1 100 2 50
i = 4	TRUE	FALSE	1 100 2 50 4 25
...			
i = 10	TRUE	TRUE	1 100 2 50 4 25 5 20 10

Рассмотренный алгоритм решает поставленную задачу, выполняя 10 итераций цикла при N=100. Таким образом, данный алгоритм будет работать быстрее, чем предыдущее алгоритмы.

Алгоритмы поиска всех делителей заданного натурального числа имеет много приложений. Например, с его помощью можно установить, является ли заданное натуральное число простым или составным. Напомним, что число, которое делится только на 1 и само на себя, называется простым; все остальные натуральные числа называются составными. В этом случае, с помощью алгоритма поиска всех делителей можно найти количество делителей, а затем провести проверку: если количество

всех делителей равно 2, то число простое, иначе составное. Данный алгоритм можно реализовать с помощью следующей программы:

```
program example9_4;
 var n, i, k: integer;
begin
 writeln('Введите целое число n'); readln(n);
 k:=2; {устанавливаем начальное количество делителей}
 for i:=2 to trunc(sqrt(N)) do
  if n mod i = 0      {проверка: i делитель n?}
   then if sqr(i)=N          {если у i нет парного делителя, то }
        then k:=k+1  else k:=k+2        {увеличиваем k на 1, иначе на 2}
 if k =2       {проверка: число простое?}
 then writeln('число простое') else writeln('число составное');
end.
```

Результат работы программы:

Значение n	*Сообщение на экране*
8	число составное
111	число простое

С помощью предложенного алгоритма можно решить и более сложную задачу. Например, дано число N. Составить программу вывода следующего за ним простого числа K. Идея решения данной задачи следующая: перебираем все натуральные числа начиная с N + 1 с шагом 1 до тех пор, пока не встретим простое число. Для решения задачи используем два цикла. Первый (внешний) – перебирает натуральные числа начиная с N+1, второй (внутренний) – определяет, является ли рассматриваемое число простым или нет. Приведенный алгоритм можно реализовать следующей программой:

```
program example9_5;
 var n, i, k: integer;
begin
 writeln('Введите целое число n'); readln(n);
 repeat       {внешний цикл }
   n:= n+1; k:=2;     {берем следующее натуральное число}
   for i:=2 to n div 2 do   {находим для него количество делителей}
    if n mod i = 0  then k:=k+1;
 until k = 2; {останавливаем перебор чисел, когда встретим простое число}
 writeln(n)
end.
```

Результат работы программы:

Значение n	*Сообщение на экране*
5	7
74	79

9.2. Алгоритм, раскладывающий натуральное число на цифры

Известно, что любое натуральное число $A = a_n a_{n-1} \ldots a_1 a_0$, где a_n, a_{n-1}, ..., a_0 - цифры числа, можно представить следующим образом:

$$A = a_n a_{n-1} \ldots a_1 a_0 = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0 =$$
$$\left(\left(\ldots\left(\left(a_n \cdot 10 + a_{n-1}\right)\cdot 10 + a_{n-2}\right)\cdot 10 \ldots\right)\cdot 10 + a_1\right)\cdot 10 + a_0.$$

Например, число 1234 можно представить как:

$$1234 = 1\cdot 10^3 + 2\cdot 10^2 + 3\cdot 10^1 + 4\cdot 10^0 = \left(\left(1\cdot 10 + 2\right)\cdot 10 + 3\right)\cdot 10 + 4.$$

Из данного представления видно, что получить последнюю цифру можно, если найти остаток от деления числа на 10. В связи с этим для разложения числа на составляющие его цифры можно использовать следующий алгоритм:

1. Находим остаток при делении числа N на 10, т.е. получаем крайнюю правую цифру числа.
2. Находим целую часть числа при делении N на 10, т.е. отделяем от числа N крайнюю правую цифру.
3. Если преобразованное N > 0, то переходим на пункт 1. Иначе число равно нулю и отделять от него больше нечего.

Предложенный алгоритм можно реализовать следующей программой:

```
program example9_6;
 var n: longint;
     b: integer;
begin
 writeln('Введите натуральное число'); readln(n);
 while n>0 do       {проверка: остались ли не отделенные цифры в числе}
  begin b:=n mod 10; write(b,' '); {получаем крайнюю правую цифру числа и выводим ее
    n:=n div 10 end;        {отделяем от исходного числа крайнюю правую цифру}
end.
```

Рассмотрим подробнее, как работает алгоритм, когда введено число a = 2456:

№ шага	Проверка условия a > 0	Значение b, a после выполнения цикла	Результат (вид экрана)
1	TRUE	b = 6, a =245	6
2	TRUE	b = 5, a = 24	6 5
3	TRUE	b = 4, a = 2	6 5 4
4	TRUE	b = 2, a = 0	6 5 4 2
5	FALSE	тело цикла не выполняется	6 5 4 2

Недостатком данного алгоритма является то, что цифры числа выводятся на экран в обратном прядке.

Данный алгоритм можно использовать для решения различных практических задач. Например, можно находить сумму цифр заданного натурального числа:

```
program example9_7;
 var n:longint;
     b, sum :integer;
begin
 writeln('Введите натуральное число'); readln(n);
 sum:=0;
 while n>0 do              {проверка: остались ли не отделенные цифры в числе}
  begin
    b:=n mod 10;           {получаем крайнюю правую цифру числа}
    sum: = sum + b;        {добавляем новую цифру к сумме цифр числа}
    n:=n div 10;           {отделяем от исходного числа крайнюю правую цифру}
  end;
 writeln(' Сумма цифр числа  =', sum );
end.
```

Результат работы программы:

Значение n	sum
0	0
123456	21

С помощью предложенного алгоритма можно решить и более сложную задачу, например, для заданного натурального числа N найти ближайшее меньшее данного число, сумма цифр которого кратна натуральному числу C.

```
program example9_8;
  var n, a: longint;   c, b, sum: integer;
begin
  writeln('Введите натуральное число n'); readln(n);
  writeln('Введите натуральное число c'); readln(c);
  repeat
    n := n - 1; a := n; {берем предыдущее натуральное число и запоминаем его копию}
    sum:=0; {находим сумму цифр числа a}
    while a>0 do
      begin  b:=a mod 10; sum: = sum + b; a:=a div 10  end;
  {останавливаем перебор чисел тогда, когда сумма цифр числа  станет кратной c}
  until sum mod c =0;
  writeln(n);
end.
```

Результат работы программы:

N	C	Ответ:
65	15	0
325	15	294

9.3. Алгоритмы нахождения наибольшего общего делителя двух натуральных чисел

Пусть даны два натуральных числа A и B. Если и A, и B одновременно делятся на число C, причем C наибольшее из всех возможных делителей, то C называется наибольшим общим делителем или НОД.

Если НОД чисел A и B равен 1, то эти числа называются взаимнопростыми.

Для нахождения НОД двух натуральных чисел можно воспользоваться алгоритмом Евклида:

1. задать два числа;
2. пока числа не равны заменять большее число разностью большего и меньшего;
3. вывести в качестве результата любое из чисел.

Данный алгоритм можно реализовать следующей программой:

```
program example9_9;
  var a, b: longint;
begin
  writeln('Введите натуральное число a'); readln(a);
  writeln('Введите натуральное число b'); readln(b);
  while a<>b do
     if a>b then a:=a-b else b:=b-a;
  writeln(a);
end.
```

Рассмотрим подробнее, как работает алгоритм для a=128 и b=160:

№ шага	Проверка условия a <>b	Проверка условия a>b	Значение a	Значение b
До входа в цикл			128	160
1	TRUE	FALSE	128	32
2	TRUE	TRUE	96	32
3	TRUE	TRUE	64	32
4	TRUE	TRUE	32	32
5	FALSE	тело цикла не выполняется		

На экран будет выведено значение 32.

Для нахождения НОД двух натуральных чисел можно воспользоваться модификацией алгоритма Евклида:

1. задать два числа;
2. пока оба числа не равны нулю заменять большее число остатком от деления большего числа на меньшее число;
3. вывести в качестве результата сумму преобразованных чисел.

Данный алгоритм можно реализовать следующей программой:

```
program example9_10;
  var a, b: longint;
begin
  writeln('Введите натуральное число a'); readln(a);
  writeln('Введите натуральное число b'); readln(b);
  while (a>0) and (b>0) do
     if a>b then a:=a mod b else b:=b mod a;
  writeln(a+b);
end.
```

Рассмотрим подробнее, как работает алгоритм для a=128 и b=160:

№ шага	Проверка условия (a >0) and (b>0)	Проверка условия a>b	Значение a	Значение b
До входа в цикл			128	160
1	TRUE	FALSE	128	32
2	TRUE	TRUE	0	32
3	FALSE	тело цикла не выполняется		

В результате на экран будет выведено значение 32.

Данный алгоритм можно использовать для нахождения наименьшего общего кратного (НОК), если воспользоваться свойством $НОК(A,B) = \frac{A \cdot B}{НОД(A,B)}$. Частный случай: для взаимнопростых чисел $НОК(A,B) = A \cdot B$, подумайте почему.

```
program example9_11;
  var a, b, nok, nod: longint;
begin
  writeln('Введите натуральное число a'); readln(a);
  writeln('Введите натуральное число b'); readln(b);
  nok:=a*b;   {первоначально записываем в nok произведение чисел A и B}
  while (a>0) and (b>0) do  {применяем алгоритм для нахождения НОД}
     if a>b then a:=a mod b  else b:=b mod a;
  nod:=a+b;  nok:=nok/nod; writeln(nok); {вычисляем НОК}
end.
```

Результат работы программы:

A	B	НОК
21	20	420
21	35	105

С помощью предложенного алгоритма можно решить и более сложную задачу, например, проверить будет ли НОД двух натуральных чисел простым числом.

```
program example9_12;
 var a, b, nod, i, k: longint;
begin
 writeln('Введите натуральное число a'); readln(a);
 writeln('Введите натуральное число b'); readln(b);
 while (a>0) and (b>0) do  {применяем алгоритм для нахождения НОД}
    if a>b then a:=a mod b
           else b:=b mod a;  nod:=a+b;
 k:=2; {применяем алгоритм нахождения количества делителей натурального числа}
 for i:=2 to trunk(sqrt(nod)) do
   if nod mod i = 0 then if i*i=nod then k:=k+1  else k:=k+2;
 if k=2 then writeln ('НОД, равный ', nod, ' является простым числом')
        else  writeln ('НОД, равный ', nod, ' является составным числом')
end.
```

Результат работы программы:

A	B	Сообщение на экране
22	33	НОД, равный 11, является простым числом
22	44	НОД, равный 22, является составным числом

9.4. Упражнения

I. Объясните, что будет напечатано программой:

1)
```
program ex1;
  var n, i: integer;
 begin
  n:=200;
  for i:=1 to n do
   if (n mod i = 0) and (i mod 10 =0)
    then write(i,'  ');
 end.
```

2)
```
program ex2;
  var n, i: integer;
 begin
  n:=128; k:=0;
  for i:=1 to n do
    if (n mod i = 0) and (i mod 2 <>0) then k:= k+1;
  writeln(k);
 end.
```

3)
```
program ex3;
  var n: longint; b, k: byte;
 begin
  n:=45654; k:=0;
  while n>0 do
   begin  b:=n mod 10;
          if b= 5 then k:=k+1;
          n:=n div 10; end;
  writeln(k);
 end.
```

4)
```
program ex4;
  var n, k: longint;
      b: byte;
 begin
  n:=45654; k:=0;
  while n>0 do
   begin  k:=k+n;
          n:=n div 10   end;
  writeln(k);
 end.
```

5)
```
program ex5;
  var a, b, x, y: longint;
 begin
  a:=124; b:=248;
  x:=a; y:=b;
  while (a>0) and (b>0) do
   if a>b then a:=a mod b
          else b:=b mod a;
  writeln(x div (a+b), y div (a+b));
end.
```

6)
```
program ex6;
  var a, b, n: longint; s:integer
 begin
  a:=1488; b:=1860; x:=a; y:=b;
  while (a>0) and (b>0) do
   if a>b then a:=a mod b else b:=b mod a;
  n:=a+b; s:=0;
  while n>0 do
    begin  s:=s+n mod 10; n:= n div 10; end;
  writeln(s);
 end.
```

II. Найти и объяснить ошибки в каждом фрагменте программы:

1) *вычисление суммы всех делителей числа N*

```
program ex1;
 var n, i: integer;
begin
 read(n); sum:=0;
 for i:=2 to n div 2  do
  if n mod i = 0 then sum:=sum+i;
 writeln(sum);
end.
```

2) *вычисление количества четных цифр числа N*

```
program ex2;
 var n, longint; b, k: byte;
begin
 readln(n); k:=0;
 while n>0 do
  begin b:=n mod 10; k:=k+1;
        n:=n div 10 end;
 writeln(k);
end.
```

3) *проверка – является ли НОД чисел a, b трехзначным числом*

```
program ex3;
 var a, b: longint;
begin
 readln(a, y);
 while (a>0) and (b>0) do
  begin
   if a>b then a:=a mod b else b:=b mod a;
   if (a+b>99) or (a+b<1000)
    then writeln ('да') else writeln ('нет');
  end;
end.
```

4) *проверка – являются ли все цифры числа N простыми*

```
program ex4;
 var n, i: integer;
begin
 read(n);
 while n>0 do
  begin
   b:=n mod 10; n:= n div 10;
   if b mod 2 = 0
    then writeln ('да') else writeln ('нет');
  end;
end.
```

III. *Для заданного натурального числа N:*

1. найти количество всех делителей;
2. найти сумму всех делителей;
3. найти наибольший делитель, не совпадающий с самим числом N;
4. вывести на экран все делители, кратные целому числу C;
5. вывести на экран все делители кратные целым числам C и D одновременно;
6. найти сумму всех делителей, не кратных целому числу C;
7. найти сумму всех делителей, кратных хотя бы одному из целых чисел C или D;
8. найти среднее арифметическое всех делителей;
9. найти среднее арифметическое всех двузначных делителей;
10. найти среднее арифметическое всех делителей, попадающих в отрезок [a, b];
11. определить является ли заданное число простым, если нет, то вывести на экран все его делители;
12. найти старшую цифру;
13. найти количество цифр;
14. найти среднее арифметическое значение цифр.

Даны два натуральных числа a и b:

15. сократить дробь вида a/b;
16. вычислить значение выражения $\frac{a}{b}+\frac{b}{a}$; результат представить в виде обыкновенной дроби, выполнив сокращение;
17. найти наибольшую цифру из старших цифр заданных чисел;
18. определить, в каком числе содержится больше значащих нулей;
19. определить, у какого числа больше делителей.

20. Даны три натуральных числа a, b и c. Найти НОД(a, b, c).

IV. *Вывести на экран все числа из отрезка [a; b]:*

1. имеющие наибольшее количество делителей;
2. имеющие наименьшее количество делителей;
3. имеющие ровно K делителей;
4. сумма делителей которых кратна натуральному числу C;
5. сумма делителей которых является простым числом;
6. в записи которых встречается цифра C;
7. в записи которых ровно K четных цифр;
8. в записи которых все цифры различны.

Дано натуральное число N. Вывести на экран:

9. предшествующее по отношению к нему простое число;
10. ближайшее большее данного число, сумма цифр которого кратна числу C;
11. ближайшее большее данного число, сумма цифр которого кратна числам C и D одновременно;
12. ближайшее меньшее данного число, сумма цифр которого кратна числу C;
13. ближайшее меньшее данного число, сумма цифр которого кратна хотя бы одному из чисел C или D;
14. ближайшее большее данного число, среди делителей которого есть число C;
15. ближайшее меньшее данного число, среди делителей которого нет числа C;

На отрезке [a, b] найти все пары соседних чисел:

16. сумма которых является составным числом;
17. произведение которых является простым числом;
18. сумма которых образует симметричное число;
19. которые являются взаимнопростыми числами;
20. наибольший общий делитель которых является простым числом.

10. ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ TURBO PASCAL

Версия TP 7.0 имеет интегрированную среду программирования, включающую в себя текстовой редактор, компилятор, компоновщик, отладчик, что позволяет писать и редактировать тексты программ, компилировать, компоновать, отлаживать и запускать программы не выходя из среды. Ее особенностями является:

- использование многих перекрывающихся окон, у которых можно менять размеры и которые можно перемещать по экрану;
- наличие развитой системы меню;
- наличие диалоговых окон;
- поддержка работы с мышью;
- многофайловый редактор, позволяющий работать с файлами до 1Мбайт, и осуществлять обмен информацией между отдельными окнами редактирования, а также использовать сведения из системы информационной помощи;
- возможность полной очистки и восстановления экрана.

Рассмотрим некоторые аспекты работы в интегрированной среде TP.

10.1. Идентификация файлов

Программы и данные, обрабатываемые интегрированной средой, хранятся на носителях информации (дисках, дискетах, флешь-картах) в виде физических файлов. В файлах могут храниться:

1. тексты исходных программ на языке Паскаль;
2. оттранслированные программы, готовые к выполнению;
3. исходные данные и результаты.

Каждый физический файл должен иметь свое уникальное имя. Оно формируется по правилам операционной системы Dos и состоит из двух частей: имени файла и расширения. Имя может содержать от 1 до 8 символов, расширение начинается с точки, за которой может следовать от 1 до 3 символов. Символы в имени и расширении могут быть написаны прописными и строчными латинскими буквами, цифрами и символами: -, _, $, #, &, !, %, (,), {, }. Прописные строчные буквы эквивалентны. Первый символ в имени файла обязательно латинская буква. Перед именем файла могут указываться имена устройств носителей информации и каталогов. Формат идентификации файла:

[устройство:\] [имя каталога\] имя файла[.расширение]

Пример: F:\prog.pas или PROG.PAS

Расширение имени файла необязательно. Оно описывает тип файла. В таблице стандартные для TP расширения:

Расширение	*Назначение файла*
.pas	Текст программы на языке TP
.bak	Предыдущая копия файла
.exe	Программа, готовая к выполнению (автономному запуску из NC, Windows)
.tpu	Готовый к выполнению модуль
.tpl	Файл библиотеки TP
.dsk	Файл указателя TP; он восстанавливает после перерыва работы с TP среду в том состоянии, в котором она была при завершении последнего сеанса работы с TP

10.2. Основные приемы работы в интегрированной среде

Для входа в интегрированную среду следует запустить исполняемый файл Turbo.exe. После запуска TP экран примет вид, приведенный на рис. 10.1. Верхняя строка экрана – главное меню, под главным меню находится окно редактора. Самая нижняя строка содержит список клавиш, которые наиболее часто используются при работе в среде. В различных ситуациях на основном экране могут появляться различные окна, например, окно подсказки (помощь), компиляции, результатов работы программы и т.д.

После появления экрана (см. рис.10.1) можно выполнять все виды работ: создавать и отлаживать программы, исправлять ошибки и т.д.

Для выхода из среды надо нажать клавиши Alt + X.

Практически все операции в среде осуществляются с помощью окон – выделенных участков поля экрана, в которых размещается определенная информация. Пользователь может открыть, закрыть окно или несколько окон, разместить их встык или с перекрытиями, перемещать по экрану и изменять размеры окон.

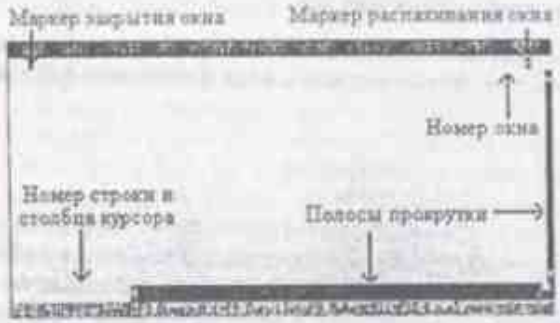


рис. 10.1. Вид экрана после вызова среды Turbo Pascal

Одновременно может быть открыто различное количество окон, но активным (доступным для выполнения тех или иных операций) может быть только одно окно. Рамка активного окна имеет белый цвет

Меню среды TP имеет древовидную структуру, состоящую из главного меню, подменю и их команд. Командами называются пункты меню, вызывающие определенные действия, а не разворачивание очередного подменю.

Переход из любого окна в главное меню осуществляется нажатием клавиши F10. При этом курсор перемещается в главное меню в виде подсветки одного из пунктов меню. К предыдущему уровню меню можно перейти с помощью клавиши Esc. Выбор требуемой команды главного или дополнительного меню осуществляется его подсветкой путем перемещения курсора с помощью мыши или клавиш управления курсором: ←, ↑, →, ↓. После выбора пункта нажатие на клавишу Enter вызывает разворачивание дополнительного меню или выполнение команды. Назначение пунктов главного меню и команды перехода в соответствующее меню второго уровня приведены в следующей таблице:

Пункт	*Назначение*	*Команда*
File	Операции с файлами, выход из системы	Alt + F
Edit	Редактирование текста в активном окне	Alt + E
Search	Поиск фрагментов текста, местоположения ошибок	Alt + S
Run	Трансляция, редактирование и запуск программы	Alt + R
Compile	Компиляция программы	Alt + C
Debug	Средства отладки программы	Alt + D
Tools	Меню инструментальных средств	Alt + T
Options	Управление параметрами компиляции, компоновки и среды Паскаля	Alt + O
Window	Управление окнами	Alt + W
Help	Обращение к системе оперативной подсказки	Alt + H

В качестве примера работы в интегрированной среде TP рассмотрим приемы изменения текущей папки, сохранения, открытия и создания копии файла, запуска программы на выполнение.

Смена папки

После запуска программы текущей является та папка, из которой было запущено приложение. Однако сохранять файлы каждому пользователю нужно в свою собственную. Для того чтобы изменить текущую папку нужно выполнить следующую последовательность действий:

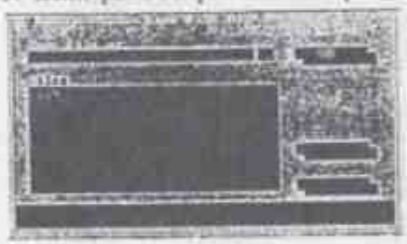
1. Нажать F10 для выхода в главное меню.
2. В главном меню выбрать команду File-Change dir... После чего откроется диалоговое окно Change Directory.

Замечание. Переход между элементами окна осуществляется с помощью клавиши табуляции Tab. Сейчас текущей считается папка C:\TEMP

3. Для смены диска нужно дважды щелкнуть левой кнопкой мыши по слову Drives или установить курсор на это слово и нажать клавишу Enter.
4. Далее нужно указать путь к требуемой папке – это делается так же как и при работе в операционной системе Windows.
5. Нажать кнопку OK чтобы ваши действия сохранились

Сохранение файла

1. Нажать F10 для выхода в главное меню.
2. В главном меню выбрать команду Save.
3. Если файл сохраняется впервые, то откроется диалоговое окно Save File As



В строке Save file as нужно указать имя файла. Если вы не укажите расширение, то TP автоматически добавит расширение pas. Если вам нужно сохранить файл с каким то другим расширением, то при определении имени файла нужно обязательно указывать и расширение.

Если же файл уже один раз был сохранен, то при повторном выполнении данной команды содержимое файла на диске автоматически обновится, а окно сохранения открываться не будет

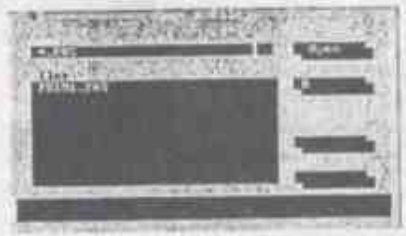
4. Нажать кнопку OK чтобы ваши действия сохранились

Создание копии файла

1. Выйти в главное меню – F10.
2. Выполнить команду Save as - откроется окно Save File As.
3. В этом окне нужно задать имя файла, в который будет сохранена копия исходного файла.
4. Нажать кнопку OK чтобы ваши действия сохранились.

При выборе некоторых команд из меню второго уровня вы видели, что рядом с ними указывалось название клавиши. Например, рядом с командой Save указано F2. Это означает, что при нажатии на клавишу F2 будет выполнена команда Save. Использование функциональных клавиш значительно ускоряет работу в интегрированной среде, поэтому их принято называть *горячими клавишами* TP.

Рассмотрим приемы работы в интегрированной среде с использованием горячих клавиш.



Открытие файла

1. Нажать клавишу F3 – откроется диалоговое окно Open a File.
2. Из предложенного списка раздела Files нужно выбрать, какой именно файл следует открыть. После чего нужно нажать кнопку Open.

Замечание. При необходимости перед открытием файла можно сменить текущую папку.

Запуск программы, разработанной в интегрированной среде TP, выполняется нажатием комбинации клавиш CTRL+F9. Но прежде чем запускать программу необходимо ее проверить, речь об этом пойдет в следующем разделе.

Полный список горячих клавиш TP с указанием их назначения приведен в следующей таблице:

Клавиши	*Функция*	*Эквивалент меню*
F1	Вызвать справочную службу	
F2	Запомнить редактируемый файл на носитель информации с тем же именем	File/ Save
F3	Загрузить текст файла в окно редактора	File/ Open
F4	Выполнить программу до строки расположения курсора	Run/ Go to cursor
F5	Раскрыть текущее окно на весь экран или обратно	Window/ Zoom
F6	Перейти в следующее окно	Window/ Next
Shift + F6	Перейти в предыдущее окно	Window/ Previous
F7	Выполнить трассировку программы (построчное выполнение программы) с заходом в подпрограммы	Run/ Trace into
F8	Выполнить трассировку программы без захода в подпрограммы	Run/ Step over
F9	Создать исполняемый файл (exe – файл)	Compile/ Make
F10	Перейти из любого окна в главное меню	
Alt + F3	Закрыть активное окно	Window/ Close
Alt + F5	Показать окно вывода результатов работы программы	Debug/ User screen
Alt + F9	Выполнить компиляцию программы, сформировав exe-файл	Compile/ Compile
Alt + X	Выход из среды TP	File/ Exit
Ctrl +F2	Сбросить отладочные средства программы (снять трассировку программы)	Run/ Program reset
Ctrl + F7	Добавить выражение или имя переменной в окно просмотра (Watch)	Debug/Add watch
Ctrl + F8	Переключить (установить или сбросить) точку прерывания	Debug/ Add breakpoint
Ctrl + F9	Запустить программу на компиляцию, компоновку и выполнение	Run/ Run
Ctrl + Del	Удалить выделенный блок текста программы	Edit/ Clear
Ctrl + Ins	Скопировать выделенный блок программы в буфер обмена	Edit/ Copy
Shift + Del	Перенести выделенный блок программы в буфер обмена	Edit/ Cut
Shift + Ins	Скопировать блок из буфера обмена в окно редактирования	Edit/ Paste

10.3. Ошибки, возникающие при разработке программ

Отладка является непременным этапом при создании любой программы, так как при ее написании обычно допускаются различные ошибки. Ошибки могут быть трех типов: синтаксические, семантические и логические.

Синтаксические ошибки

Синтаксические ошибки возникают в результате нарушения правил написания команд языка, и выявляются на этапе компиляции программы. При выявлении ошибки компилятор создает сообщение о ее характере, а курсор указывает место в тексте, где эта ошибка обнаружена. Для обнаружения синтаксических ошибок необ-

ходимо нажать комбинацию клавиш Alt+F9. Наиболее часто встречающиеся синтаксические ошибки приведены в следующей таблице:

Код	*Сообщение об ошибке*	*Причины возникновения*
2	*Identifier expected* (Не указан идентификатор)	1) Пропущен идентификатор; 2) Использование в качестве идентификатора служебного слова.
3	*Unknown identifier* – (Неизвестный идентификатор)	1) Использование переменных, не описанных в разделе var; 2) Ошибочное написание служебных слов.
4	*Duplicate identifier* (Повторный идентификатор)	Повторное описание одного и того же идентификатора (переменной, процедуры или функции, созданной пользователем).
5	*Syntax error* (Синтаксическая ошибка)	1) Использование недопустимого символа; 2) Пропущен апостроф при формировании строковой константы.
8	String constant exceeds line (Строковая константа превышает допустимые размеры)	Пропущен апостроф в конце строковой константы.
10	*Unexpected end of file* – (Отсутствует конец файла программы)	1) Отсутствует end, завершающий программу; 2) Не были закончены комментарии.
12	*Type identifier expected* (Отсутствует идентификатор типа)	Не указан тип идентификатора.
21	*Error in type* (Ошибка в определении типа)	Объявление типа не может начинаться с этого символа.
26	*Type mismatch* (Несовместимость типов)	1) Несовместимы типы переменной и выражения в операторе присваивания; 2) Несовместимы типы фактического и формального параметров в обращении к процедуре или функции; 3) Тип выражения не совместим с типом индекса при индексировании массива; 4) Несовместимы типы операндов в выражении.
36	*Begin expected* (Отсутствует begin)	Возможно в вашей программе неодинаковое количество операторов BEGIN и END
37	*End expected* (Отсутствует end)	Возможно в вашей программе неодинаковое количество операторов BEGIN и END
41	*Operand types do not match operator* (Типы операндов не соответствуют операции)	Данная операция не может быть применена к указанным операндам, например, 10.7 div 2.
42	*Error in expression* *(Ошибка в выражении)*	1) Данный символ не может участвовать в выражении указанным образом. 2) Забыли указать операцию между двумя операндами.
50	*Do expected* (Отсутствует do)	Пропущено служебное слово do.
54	*OF expected* (Отсутствует OF)	Пропущено служебное слово of.
57	*THEN expected* (Отсутствует THEN)	Пропущено служебное слово then.
58	*TO or DOWNTO expected* (Отсутствует TO или DOWNTO)	Пропущено служебное слово to или downto в операторе цикла for.

62	*Division by zero* (Деление на ноль)	Предшествующая операция пытается выполнить деление на ноль.
64	*Cannot Read or Write variables of this type* (Нет возможности считать или записать переменные данного типа).	1) Процедуры READ и READLN могут считывать переменные символьного, целого, действительного и строкового типов. 2) Процедуры WRITE и WRITELN могут выводить переменные символьного, целого, действительного, булевского и строкового типов.
76	*Constant out of range* (Константа нарушает границы)	1) Выход индекса массива за его границы; 2) Переменной присваивается значение, выходящее за границы, допустимые для типа этой переменной; 3) В качестве фактического параметра процедуре или функции передается константа, выходящая за границы, допустимые для типа соответствующего формального параметра.
85-95	*... expected* (Отсутствует символ, который указан на месте многоточия)	Пропущен символ, указанный вместо многоточия
96	*Too many variables* (Слишком много переменных).	а) Размер глобальных переменных, описанных в программе или программном модуле, превышают допустимый размер - 64 Кбайт. б) Размер локальных переменных, описанных в процедуре или функции, не может превышать допустимый размер - 64 Кбайт.
97	*Invalid FOR control variable* (Неправильный параметр цикла оператора FOR)	Параметр цикла оператора FOR не является порядковым типом
113	*Error in statement* (Ошибка в операторе)	Недопустимо использовать данный символ первым символом в операторе
140	*Invalid floating-point operation* (Недопустимая операция с плавающей запятой)	При выполнении операции с плавающей запятой произошло переполнение или деление на ноль.

Семантические ошибки

Семантические ошибки возникают в результате использования недопустимых значений параметров (например, попытки записать действительное значение в целочисленную переменную) или выполнения недопустимых действий над параметрами (например, деление на ноль). Выявляются эти ошибки интегрированной средой на этапе выполнения программы, о чем также выдается сообщение. После этого сообщения программа завершает свою работу. Обобщенное сообщение выглядит следующим образом:

Runtime error nnn at xxxx:yyyy
(Ошибка периода исполнения nnn по адресу xxxx:yyyy)

где nnn - номер ошибки; xxxx:yyyy - адрес (сегмент:смещение).

Подробно с типами ошибок можно ознакомиться в справочной системе интегрированной среды. Для это, нужно выполнить следующую последовательность действий F10/ Help/ Index/ RunError. Откроется справочное окно, в котором можно изучить коды ошибок и причины их возникновения.

Логические ошибки

Логические ошибки возникают в связи с ошибочной реализацией алгоритма. Эти ошибки не приводят к прерыванию программы, но при этом выдаются неверные результаты. Отладка таких ошибок зависит от качества тестовых примеров, составленных вами.

Мы рекомендуем к каждой задаче составить набор примеров, в которых указаны входные значения и определены выходные значения, просчитанные аналитически без участия программы. Количество тестовых примеров может быть различным в зависимости от сложности программы. Чем сложнее задача, тем больше разрабатывается тестов, учитывающих все возможные общие и частные случаи. Если ваша программа правильно дала ответы на все тесты, то можно говорить о корректности работы программы. В противном случае надо устранять ошибки в логической структуре программы.

ЛИТЕРАТУРА

1. *Бабушкина Н.А., Бушмелева Н.А., Окулов С.М., Черных С.Ю.* Практикум по Турбо Паскалю: Учебное пособие по курсам «Информатика и вычислительная техника», «Основы программирования». – М.:АБФ, 1998.
2. *Епанешников А.М., Епанешников В.А.* Программирование в среде Turbo Pascal 7.0. – М.: Машиностроение, 1994.
3. Задачи по программированию/ С.М. Окулов, Т.В. Ашихмина, Н.А. Бушмелева и др.; Под. Ред. С.М.Окулова. - М.: БИНОМ. Лаборатория знаний, 2006.
4. *Климова Л.М.* PASCAL 7.0. Практическое программирование. Решение типовых задач. – М.: КУДИЦ-ОБРАЗ, 2000.
5. *Окулов С.М.* Основы программирования. – 3-е изд. – М.: БИНОМ. Лаборатория знаний, 2006.
6. *Фаронов В.В.* Turbo Pascal: Учебное пособие. – СПб.: Питер, 2007.

Содержание