Министерство образования и науки Российской Федерации
ГБОУ Лицей № 533
«Образовательный комплекс «Малая Охта»
Красногвардейского района Санкт-Петербурга

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PASCAL

Учебно-методическое пособие
по предмету «Информатика и ИКТ»
для учащихся 10-11 классов лицеев и гимназий

Санкт-Петербург
2015

Составила: А. Г. Тамаревская, учитель информатики высшей квалификационной категории Лицея № 533 Санкт-Петербурга

Рецензенты: Т. А. Андреева, м. н. с. ИСИ СО РАН, преподаватель НГУ,
Т. Г. Чурина, к. ф.-м. н., доцент НГУ

Оглавление

Вместо предисловия

В данном учебном пособии излагается материал для изучения основ программирования на языке программирования Pascal. Рассмотрены основные темы, изучаемые в средней школе. Даны рекомендации по написанию, отладке и тестировании программ.

Материал снабжен
🗎 множеством примеров,
разбором типовых алгоритмов и большого количества задач.

В конце каждой главы предлагаются:
🕮 вопросы для повторения изученного материала
💻 задания для самостоятельной работы, которые необходимо выполнить на компьютере.

Решения задач в данной версии книги (2015) приведены в формате, пригодном для сдачи в тестирующую систему.

Основные элементы программивания

Большинство программ создаются для решения какой-либо задачи. В процессе решения задачи на компьютере пользователю нужно ввести обрабатываемые данные, указать, как их обрабатывать, задать способ вывода полученных результатов. Поэтому как программист вы должны знать:

- как ввести информацию в память (ввод);
- как хранить информацию в памяти (данные);
- как указать правильные команды для обработки данных (операции);
- как передать обратно данные из программы пользователю (вывод).

Вы должны упорядочить команды таким образом, чтобы:

- некоторые из них выполнялись только в том случае, если соблюдается некоторое условие или ряд условий (условное выполнение);
- другие выполнялись повторно некоторое количество раз (циклы);
- третьи выделялись в отдельные части, которые могут быть неоднократно выполнены в разных местах программы (подпрограммы).

Таким образом, как программист вы должны уметь использовать семь основных элементов программирования: ввод, данные, операции, вывод, условное выполнение, циклы и подпрограммы и на их основе строить программы.

Этот список не является исчерпывающим, однако он содержит те элементы, которые обычно присущи всем программам (и процедурным языкам программирования). Многие языки программирования, в том числе и Pascal, имеют еще дополнительные средства, которые вы изучите далее самостоятельно. Ниже дается краткое описание каждого элемента.

Ввод означает считывание значений, поступающих с клавиатуры, с диска или из порта ввода-вывода.

Данные – это константы, переменные и структуры, содержащие числа (целые и вещественные), текст (символы и строки) или адреса (переменных или структур).

Операции осуществляют присваивание значений, их комбинирование (сложение, деление и т.д.) и сравнение значений (равные, неравные и т.д.).

Вывод означаетзапись информации на экран, на диск или в порт ввода-вывода.

Условное выполнение предполагает выполнение набора команд в случае, если удовлетворяется (является истинным) некоторое условие (если это условие не удовлетворяется, то эти команды пропускаются или же выполняется другой набор команд).

Благодаря ***циклам*** некоторый набор команд выполняется повторно или фиксированное количество раз, или пока является истинным некоторое условие, или пока некоторое условие не стало истинным.

Подпрограмма представляет собой набор команд, который имеет имя и может быть неоднократно вызван из любого места программы по его имени.

В данном пособии мы рассмотрим, как эти элементы реализуются в языке программирования Pascal.

Исторические сведения о языке Pascal

В конце 1968 года профессор Никлаус Вирт и его сотрудники из швейцарского федерального института технологии в Цюрихе разработали первую версию языка Pascal. Спустя два года – первый вариант компилятора. В 1971 году Вирт выпустил описание своего языка.

Turbo Pascal появился на рынке программных продуктов в 1983 году и совершил революцию в программировании. До этих пор предпочтение отдавалось Бейсику – простому, дешевому и легко усваиваемому. Pascal же был аппаратно зависимым, дорогим и сложным в обращении. С появлением Turbo Pascal положение меняется. Он состоит из языка программирования и среды программирования, которая создает удобства в работе.

Первая версия использовалась не очень долго – появилась в 1983 году, а уже в 1984 году ее заменила вторая версия, которая получила широкое распространение. К осени 1985 года появляется третья версия, более удобная в работе (быстрее работает компилятор и редактор, возможен вызов MS-DOS из программы). Четвертая версия (1988 год) – появилась новая среда, компилятор стал встроенным. Осенью того же года – пятая версия, у которой еще больше развита среда и появился встроенный отладчик. А в 1989 году появилась версия 5.5, позволившая перейти к объектно-ориентированному программированию. Шестая версия уже обеспечивала многооконный и многофайловый режим работы, использование «мыши», применение объектно-ориентированного программирования, обладала встроенным ассемблером и имела другие возможности.

В 1992 году фирма Borland International выпустила два пакета программирования на языке Pascal – это Borland Pascal 7.0 и Turbo Pascal 7.0.

Пакет Turbo Pascal использует новейшие достижения в программировании. Язык этой версии обладает широкими возможностями, имеет большую библиотеку модулей. Среда программирования позволяет создавать тексты программ, компилировать их, находить и исправлять ошибки, компоновать программы из отдельных частей (модулей), отлаживать и выполнять отлаженную программу.

Глава 1. Начальные сведения

1.1. Пример учебной программы

Создадим первую учебную программу вычисления суммы двух целых чисел. Сценарий взаимодействия человека и компьютера при решении данной задачи можно предложить следующий. Компьютер запрашивает у человека значение первого числа, человек набирает значение на клавиатуре и нажимает клавишу [Enter], компьютер считывает введенное число и записывает в память под именем a. Затем компьютер запрашивает значение второго целого числа, считывает его и записывает в память под именем b. После этого компьютер выполняет сложение чисел a и b, записывает результат в память под именем sum, выводит на экран сообщение 'Сумма чисел = ' и значение величины sum.

```
Program example;                          {заголовок программы}
Var a, b, sum: integer;                   {переменные a, b, sum – целые}
Begin                                     {начало программы}
   write ('введите число a: ');           {вывод запроса на экран}
   readln (a);                            {ввод значения a с клавиатуры}
   write ('введите число b: ');           {вывод запроса на экран}
   readln (b);                            {ввод значения b с клавиатуры}
   sum := a + b;                          {вычисление переменной sum }
   writeln ('Сумма чисел = ', sum);       {вывод ответа на экран}
End.                                      {конец программы}
```

Особенности работы в тестирующей системе: в формулировке задания обязательным образом описан формат входных и выходных данных, а Ваша программа должна считать эти данные, обработать и вывести результат в установленном формате. Это означает, что вывод запроса на экран (строка 4 в примере) не требуется. А вывод ответа на экран (строка 9 в примере) необходимо выполнять строго в соответствии с тем, что указано в задаче (чаще всего без каких-либо поясняющих надписей).

1.2. Алфавит языка

Алфавит языка состоит из *букв латинского алфавита* (русский алфавит используется только для задания символьных и строковых величин):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h I j k l m n o p q r s t u v w x y z
и знак подчеркивания (_);
арабских цифр:
0 1 2 3 4 5 6 7 8 9;
специальных символов, используемых как:

- знаки операций: + - * / = > < ;
- знаки пунктуации:
  [ ] выделение индексов массивов, элементов множеств;
  { } скобки комментариев;
  ( ) выделение выражений, списков параметров;
  ' выделение символа или строковой константы;
  . конец программы, отделение целой части от дробной;
  , разделение элементов в списке;
  ; разделение операторов и объявлений;
  : отделение переменной или константы от типа;

= отделение константы от ее значения;
обозначение символа по его коду и другие.

Комбинации специальных символов могут образовывать *составные символы*:

<> не равно;	:= присваивание;
<= меньше или равно;	.. разделение границ диапазона;
>= больше или равно;	(* *) альтернатива { }.

В программе эти пары символов нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария.

1.3. Слова в Pascal. Идентификаторы

Неделимые последовательности знаков алфавита образуют ***слова***, отделенные друг от друга разделителями и несущие определенный смысл в программе. Разделителем могут служить пробел, символ конца строки, комментарий. Набор слов, используемый в языке Pascal, можно разделить на три группы: зарезервированные слова, стандартные идентификаторы и идентификаторы пользователя.

Зарезервированные слова составляют основу языка Pascal и имеют строго фиксированное написание. Их нельзя использовать в качестве имен, во всех программах они имеют одинаковый смысл. К ним относятся следующие идентификаторы:

`and`	логическое И	`of`	из
`array`	массив	`or`	логическое ИЛИ
`begin`	начало блока	`procedure`	процедура
`case`	вариант	`program`	программа
`const`	константа	`repeat`	повторять
`div`	деление нацело	`shl`	сдвиг битов влево
`do`	выполнять	`shr`	сдвиг битов вправо
`downto`	уменьшить до	`string`	строка
`else`	иначе	`then`	то
`end`	конец блока	`to`	увеличивая
`for`	для	`type`	тип
`function`	функция	`until`	до
`if`	если	`Var`	переменная
`mod`	остаток от деления	`while`	пока
`not`	логическое НЕ	`xor`	исключающее ИЛИ

Идентификаторы в Pascal используются для именования констант, типов, переменных, процедур, функций, модулей, программ и полей в записях. Идентификаторы могут иметь произвольную длину, но значащими являются только первые 63 символа.

Идентификатор всегда начинается буквой (или знаком подчёркивания), за которой следуют буквы и цифры. Пробелы и специальные символы алфавита не могут входить в идентификатор.

При написании идентификаторов могут быть использованы как прописные, так и строчные буквы. Компилятор их не различает.

Примеры написания идентификаторов:

Правильно	*Неправильно*
Name	My Name
WorkPhone	1_WorkPhone
HousePhone	Unhead-of

Стандартные идентификаторы используются для именования процедур, функций, типов данных, констан, директив, входящих в Pascal. Например: `integer`, `sin`, `cos`, `ln`, `sqr`, `sqrt`, `read`, `readln`, `write`, `writeln`.

Для обозначения констант, переменных, процедур и функций, определенных самим программистом, применяются ***идентификаторы пользователя***.

Группа слов, имеющая некоторый смысл, называется словосочетанием. В языке программирования словосочетание, состоящее из слов и символов и задающее правило вычисления некоторого значение, называется ***выражением***. Минимальная конструкция языка, представляющая собой законченную мысль, есть предложение. Если предложение языка программирования задает полное описание некоторого действия, которое необходимо выполнить, оно называется ***оператором***. Предложение, описывающее структуру и организацию данных, называется ***описанием***.

1.4. Комментарии и директивы компилятора

В любом месте программы, где разрешен пробел, можно записать пояснительный текст – ***комментарий***. Он не обрабатывается компилятором и не включается в исполняемый exe-файл. Комментарий может содержать любые комбинации латинских и русских букв, цифр и других символов. Ограничений на длину комментария нет, он может занимать несколько строк.

Текст комментария ограничен символами **{ }** или **(* *)**:

{`Текст комментария`**}** или **(***`Текст комментария`***)**.

Следует знать:

- ограничители комментария удобно использовать при отладке программы для временного исключения группы операторов, которая в этом случае будет восприниматься как комментарий и, следовательно, не будет выполняться;
- комментарии необходимо отличать от ***директив компилятора***, которые используются программистом для управления режимами компиляции. Директивы, как и комментарии, заключаются в фигурные скобки, но имеют отличительный признак в виде символа **$**. По умолчанию они находятся в состоянии, гарантирующем минимальный объем исполнимого кода и минимум времени компиляции. Несколько директив компилятора могут находиться в одной строке. Часто используются следующие директивы:
    - `{$R+}` – проверять выход за границы диапазонов;
    - `{$I-}` – отмена контроля операций ввода/вывода;
    - `{$F+}` – формировать дальний тип вызова процедур и функций.

1.5. Структура программы

Программа состоит из необязательного заголовка и блока.

В **заголовок** входят: служебное слово `Program` и идентификатор (имя) программы.

Тело программы (блок) состоит из шести разделов:

- раздел меток;
- раздел констант;
- раздел типов;
- раздел переменных;
- раздел процедур и функций;
- раздел операторов.

Любой из разделов (кроме раздела операторов) может отсутствовать.

Раздел описания констант

`const` (служебное слово)

Описание каждой константы содержит идентификатор константы, знак равенства, и ее значение. Например:

```
e = 2.718281828;
```

Раздел описания типов

type (служебное слово)

Если возникает необходимость в создании своих типов данных, то это осуществляется в разделе описания типов следующим образом: за идентификатором типа записывается знак равенства, и далее в круглых скобках через запятую перечисляются те значения, которые будет принимать переменная данного типа

```
seasons = (spring, summer, autumn, winter);
```

Интервальные типы: можно объявить новый тип так, что он будет частью упорядоченного интервала

```
Days = 1 .. 31;
Month = 1 .. 12;
z = 'a' .. 'w';
```

Раздел описания переменных

Var (служебное слово)

В данном разделе должны быть перечислены все переменные, используемые в программе, с указанием их типов. Идентификаторы переменных одного типа можно записывать через запятую.

```
i, kl, l: integer;
a, b: real;
Holydays: seasons;
```

В качестве совета можно отметить следующее: если назначение переменных похоже и они имеют один и тот же тип, то лучше описать их в одной и той же строке; если назначение переменных совершенно различно, то лучше описывать их в разных строках, даже если они имеют один и тот же тип.

Раздел процедур и функций

Сам раздел не имеет служебного слова. В разделе может находиться несколько процедур или/и функций, но каждая процедура начинается со служебного слова **Procedure**, каждая функция – со слова **Function**.

Процедуры и функции по своей структуре подобны программам.

Раздел операторов (обязательный)

```
begin
   <тело программы>
end.
```

1.6. Типы данных и их описание

1.6.1. Данные. Константы, переменные

Данные – общее понятие всего того, с чем работает компьютер. В аппаратуре все данные представляются как последовательности двоичных цифр (разрядов).

Данные разделяются на две группы – константы и переменные.

Константами называются элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы *не изменяются*.

Переменными называются величины, которые *могут менять* свои значения в процессе выполнения программы.

В языке Pascal существует три вида ***констант***:

- неименованные константы;
- именованные нетипизированные константы;
- именованные типизированные константы.

Неименованные константы не имеют имен, поэтому их не надо описывать.

Тип неименованной константы определяется автоматически, по умолчанию:

- любая последовательность цифр (возможно, предваряемая знаком "-" или "+" или разбиваемая одной точкой) воспринимается компилятором как неименованная константа – число (целое или вещественное);
- любая последовательность символов, заключенная в апострофы, воспринимается как неименованная константа – строка;
- любая последовательность целых чисел либо символов через запятую, обрамленная квадратными скобками, воспринимается как неименованная константа – множество.

Кроме того, существуют две специальные константы `true` и `false`, относящиеся к логическому типу данных.

Примерами использования неименованных констант могут послужить следующие операторы:

```
int1 := -10;
real2 := 12.075 + x;
char3 := 'z';
string4 := 'abc' + string44;
boolean6 := true;
```

Именованные константы, как следует из их названия, должны иметь имя. Стало быть, эти имена необходимо сообщить компилятору, то есть описать в специальном разделе `const`.

Если не указывать тип константы, то по ее внешнему виду компилятор сам определит, к какому (базовому) типу ее отнести. Любую уже описанную константу можно использовать при объявлении других констант, переменных и типов данных. Вот несколько примеров описания ***нетипизированных именованных*** констант:

```
const n = -10;
      m = 1000000000;
      mmm = n*100;
      x = 2.5;
      c = 'z';
      s = 'string';
      b = true;
```

Типизированные именованные константы представляют собой переменные (!) с начальным значением, которое к моменту старта программы уже известно. Следовательно, во-первых, типизированные константы нельзя использовать для определения других констант, типов данных и переменных, а во-вторых, их значения можно изменять в процессе работы программы.

Описание типизированных констант производится по следующему шаблону:

const <имя_константы> : <тип_константы> = <начальное_значение>;

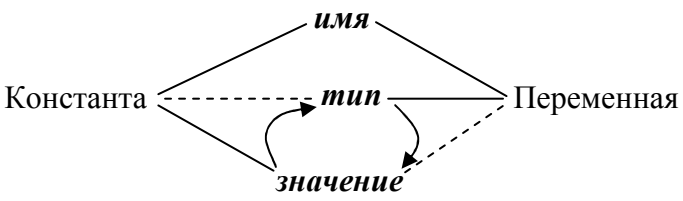
Из приведенных ниже примеров видно, как это сделать:

```
const n: integer = -10;
      x: real = 2.5;
      c: char = 'z';
      b: boolean = true;
```

1.6.2. Основные характеристики данных

Основными характеристиками и констант, и переменных в Pascal являются ***имя*** (некоторый идентификатор, задаваемый программистом), ***тип*** и ***значение***.

И константы, и переменные – это некоторые ячейки памяти. Когда в программе мы обращаемся к константам и переменным, то фактически имеем в виду те значения, которые в данный момент находятся в этих ячейках памяти. Преобразование двоичного кода, хранящегося в ячейках памяти, осуществляется в соответствии с типом данных.



При определении *константы* в явном виде (сплошные линии на схеме) задаются ее имя и значение, а тип определяется неявно (пунктирная линия) по значению. Например:

```
Const G = 9.8;
      R = '&';
      L = true;
```

При описании *переменной* в явном виде задаются ее имя и тип, а значение вводится или присваивается в ходе выполнения программы. Например:

```
Var A: real;
    B: char;
```

Каждая переменная и константа принадлежит к определенному ***типу данных***. Любой тип данных определяет ***множество значений***, которые может принимать величина этого типа, и те ***операции***, которые можно применять к величинам этого типа.

1.6.3. Классификация типов данных

Все типы данных, используемые в Turbo Pascal, можно разделить на две большие группы: скалярные (простые) и структурированные (составные).

Скалярные типы в свою очередь подразделяются на стандартные и пользовательские (перечисляемый и интервальный).

К ***стандартным*** типам относятся: целочисленные, вещественные, символьный, логический и указатели (последние мы рассматривать не будем).

Структурированные типы имеют в своей основе скалярные типы данных. К структурированным типам относятся: строки, массивы, множества, записи и файлы (из них мы рассмотрим только первые два).

Целочисленные типы, символьный, логический и пользовательские типы данных (перечисляемый и интервальный) образуют группу так называемых ***порядковых*** типов.

1.6.4. Целочисленные типы данных

Тип	Диапазон значений	Объем памяти, байт
`Shortint`	-128 … 127	1
`Integer`	-32768 … 32767	2
`Longint`	-2147483648 … 2147483647	4
`Byte`	0 … 255	1
`Word`	0 … 65535	2

Операции с величинами целого типа: сложение (+), вычитание (-), умножение (*), нахождение целой части деления (`div`), нахождение остатка от деления (`mod`), операции сравнения. Так как целый тип данных относится к порядковому типу, то можно применять стандартные функции `Ord`, `Succ`, `Pred`.

1.6.5. Вещественные типы данных

Вещественные числа могут изображаться в естественной и экспоненциальной форме. Число 234000 записано в естественной форме. Мы можем записать его и так $2.34 \cdot 10^5$. Запись данного числа в экспоненциальной форме выглядит следующим образом: 2.34E+5.

Вещественные числа записываются в виде мантиссы (дробной части числа), умноженной на порядок (степень десяти, обозначается буквой E). Знак и число, стоящие после символа E, указывают, на сколько знаков вправо или влево необходимо сместить десятичную запятую.

Например:

65470 = 6.547E+4
3.1415 = 3.1415E+0
0.0028 = 2.8E-3

Тип	Диапазон возможных значений	Точность, знаков	Объем памяти, байт
`Real`	2.9E-39 .. 1.7E38	11-12	6
`Single`	1.5E-45 .. 3.4E38	7-8	4
`Double`	5.0E-324 .. 1.7E308	15-16	8
`Extended`	3.4E-4932 .. 1.1E4932	19-20	10
`Comp`	-9.2E18 .. 9.2E18	19-20	8

Допустимые арифметические операции: сложение (+), вычитание (-), умножение (*), деление (/) и операции сравнения.

На вещественных типах данных нет отношения порядка, поэтому операцией сравнения «равно» (=) следует пользоваться с большой осторожностью.

При разработке программы необходимо учитывать неточность вычислений. Если `a = b` *при условии абсолютно точных вычислений, то это совершенно не обязательно так, когда вычисления неточны. Вполне может произойти ситуация, когда из-за погрешностей вычислений переменные* `a` *и* `b` *получат очень близкие, но все-таки не равные значения, хотя при абсолютной точности вычислений они должны были бы получить равные значения. В результате программа выполнит совсем другой набор действий и отработает неправильно.*

Во избежание таких случаев сравнение вещественных чисел обычно проводят с некоторой точностью. В частности, вместо условия `a = b` *обычно используется условие* `Abs(a - b) < e`*, где* `e` *– точность сравнения. Обычно* `e` *полагают достаточно малым, например,* `e = 0.00000001`*. В этом случае Ваша программа окажется гораздо более защищенной от погрешностей вычислений.*

Использование типов `single`, `double`, `extended`, `comp` возможно только при включенной директиве компилятора {$N+}. По умолчанию она находится в выключенном состоянии.

Основные факторы, влияющие на выбор типа переменной:

- диапазон значений в вещественном типе шире, чем в целом типе;
- операции в целом типе выполняются быстрее, чем в вещественном;
- операции в вещественном типе выполняются с некоторой погрешностью, а в целом типе приблизительный результат дает только операция деления (/). Поэтому, например, переменные-счетчики должны иметь целый тип.

1.6.6. Символьный тип данных

Значением символьного типа является множество всех символов ПК – цифры, буквы, знаки операций, специальные символы и т.д.

Каждому символу приписывается целое число в диапазоне 0..255. Это число служит кодом внутреннего представления символа.

Для кодировки используется код ASCII (American Standart Code for Information Interchange – американский стандартный код для обмена информацией).

Идентификатором символьного типа является зарезервированное слово Char, например:

```
Var c, ch: char;
```

Символьные значения можно вводить и выводить, присваивать, сравнивать. Из двух символов большим считается тот, код которого больше (в таблице ASCII он встречается позже).

В программе значения переменных и констант символьного типа должны быть заключены в *апострофы*. Например:

```
ch := '*';    a := '3';    Letter := 'G';
```

Кроме того, значения можно задать указанием кода:

```
k := #65;
```

1.6.7. Строковый тип данных

Значением строкового типа является последовательность символов.

Идентификатором строкового типа является зарезервированное слово `String`, например:

```
Var st, str: string;
str2: string[15];
```

Переменная, объявленная как переменная типа `String`, содержит от 0 до N символов, но не может превышать 255. Длину строки можно указать в квадратных скобках рядом со служебным словом `String`. В том случае, если длина строки не указана, она принимается максимально возможной, то есть 255 символов.

1.6.8. Логический тип данных

Переменные логического типа описываются с помощью идентификатора `Boolean`. Могут принимать одно из двух значений: `False` (ложь) или `True` (истина), размер выделяемой памяти – 1 байт. Тип является перечислимым, поэтому `False` < `True`, `Ord` (`False`) = 0, `Ord` (`True`) = 1, `Succ` (`False`) = `True`, `Pred` (`True`) = `False`.

Над переменными логического типа в Pascal допустимы операции дизъюнкция (`or`), конъюнкция (`and`), отрицание (`not`), исключающее «или» – сложение по модулю два (`xor`).

Величины логического типа можно присваивать, выводить, но нельзя вводить оператором `read`. Например,

```
X := True;
Y := 5 > 3;
If not X then …
```

1.6.9. Пользовательские типы данных

В языке Pascal имеются два дополнительных пользовательских порядковых типа:

- интервальный (ограниченный) тип или диапазон;
- перечисляемый тип.

Интервальный тип задается своим минимальным и максимальным значениями и может быть опеределен на основе любого порядкового типа. Например:

```
Type month = 1..12; {номер месяца может принимать значения от 1 до 12}
     symbol = 'a'..'z';   {буквы латинского алфавита}
```

Перечисляемый тип ограничен больше, он задается перечислением своих значений. Например, в виде строковых констант:

```
Type color = (red, blue, green, black);
```

В этом примере создан новый (нестандартный) тип данных `color`. Переменные этого типа могут принимать всего 4 значения: `red, blue, green` и `black`.

🕮 Вопросы для повторения:

1. В какой форме изображаются вещественные числа в Pascal?
2. Из каких частей состоит программа на Pascal?
3. Какие разделы могут входить в блок программы?
4. Какие разделы не являются обязательными при написании программы?
5. Как в тексте программы определить, что начался раздел меток?
6. Как в тексте программы определить, что закончился раздел констант?
7. Назовите стандартные типы данных.

💻 Задания для самостоятельной работы:

1. Переведите числа в естественную форму записи:
   1) 4.09E+02 3) 7.802E+01 5) 3.3E-02
   2) 2.97E-03 4) 5.29E+00 6) 2.03E+05
2. Переведите числа в экспоненциальную форму записи:
   1) 234000 3) 0.0045 5) 678
   2) 7.302 4) 4090 6) 0.0306
3. Какое из чисел больше: а) 4.67E-01 б) 0.0467.
4. Какое из чисел меньше: а) 9.21E+04 б) 9210
5. К каким типам можно отнести перечисленные ниже данные:

1) 200	6) -17	11) 2305
2) 5.89	7) 56890	12) -56890
3) -32767	8) -32769	13) 2147483648
4) {	9) 7	14) Привет
5) 1945год	10) -31009	15) `byte`

1.7. Выражения и операции

Выражение задает порядок выполнения действий над элементами данных и состоит из ***операндов*** (констант, переменных всех типов, обращений к функциям), круглых скобок и знаков ***операций***. Операции определяют действия, которые надо выполнить над операндами. В простейшем случае выражение может состоять из одной переменной или константы.

Операции в Turbo Pascal подразделяются на арифметические, отношения, логические (булевские), строковые и т.д. Выражения соответственно называются арифметическими, отношениями, булевскими, строковыми и т.д. в зависимости от того, какого типа операнды и операции используются.

Тип значения, вычисляемого с помощью выражения, определяется типом его операндов и выполняемыми над ними операциями.

Различают бинарные операции (выполняются над двумя операндами) и унарные (над одним операндом). Бинарные операции записываются в обычной математической форме, знак унарной операции предшествует операнду.

Например:

`-X` – унарная операция, `X + Y` – бинарная операция.

Порядок выполнения операций в выражении определяется правилами приоритета (см. 1.7.4).

1.7.1. Арифметические выражения и операции

Арифметическим (АВ) называется выражение, составленное из операндов арифметического типа и использующее только знаки арифметических операций и круглые скобки. Арифметические операции выполняют арифметические действия в выражениях над значениями операндов целочисленных и вещественных типов.

В языке Pascal используются следующие арифметические операции:

Операция	Действие	Тип операндов	Тип результата
Бинарные			
+	Сложение	целый вещественный	целый вещественный
-	Вычитание	целый вещественный	целый вещественный
*	Умножение	целый вещественный	целый вещественный
/	Деление	целый вещественный	вещественный
`div`	Целочисленное деление	целый	целый
`mod`	Остаток от деления	целый	целый
`shl`	Сдвиг влево	целый	целый
`shr`	Сдвиг вправо	целый	целый
Унарные			
+	Сохранение знака	целый вещественный	целый вещественный
-	Отрицание знака	целый вещественный	целый вещественный

Операции сложения, вычитания, умножения, деления (*/*) допускают совместное использование операндов вещественного и целого типов, при этом результат операции – вещественный.

Целочисленное деление (**`div`**) отличается от обычного деления тем, что вычисляет целую часть от частного, дробная часть отбрасывается. Если делимое меньше делителя, результат равен нулю.

Деление по модулю (**`mod`**) вычисляет остаток, полученный при выполнении целочисленного деления. Например:

выражение	результат
11 `div` 5	2
10 `div` 3	3
10 `mod` 3	1
14 `mod` 5	4

Для целого *b*, не равного 0, и любого целого *a* справедливы тождества

```
a div b = – ((–a) div b) = – (a div (–b)) = (–a) div (–b);
a mod b = – ((–a) mod b) = a mod (–b) = – ((–a) mod (–b)).
```

Такое определение целочисленного деления для отрицательных чисел, однако, оказывается не всегда удобным при решении задач.

Сдвиг влево (**`k shl n`**) вычисляет значение, полученное путем сдвига на *n* позиций влево представленного в двоичной форме числа *k*.

Сдвиг вправо (**`k shr n`**) выполняется аналогично, но вправо.

В языке Pascal нет возведения в степень.

Если степень целая, её заменяют умножением:

$$a^3 \quad \leftrightarrow \quad a*a*a\,.$$

Для возведения числа в произвольную степень используется следующее соотношение:

$$x^y = e^{y\ln x}\,.\ \text{Тогда}\ x^y = \exp\left(y*\ln(x)\right).$$

🗎 **Рассмотрим пример.**

Требуется вычислить $\sqrt[3]{a-b}$ $(a \neq b)$, то есть надо вычислить $(a-b)^{\frac{1}{3}}$. Выражение $\exp(1/3 * \ln(abs(a-b)))$ дает абсолютную величину искомого результата. Требуемый его знак получаем, умножая выражение на дробь $(a-b)/abs(a-b)$. Тип определяет внутримашинное представление числа.

🗎 **Рассмотрите** внимательно **примеры** использования арифметических действий.

ПРАВИЛЬНО:

```
 Var a, b: integer;
        r: integer;
        s: integer;
    …
     r := a div b;      {a = 7, b = 2, r = 3}
     r := a mod b;      { a = 7, b = 2, r = 1}
     s := a * b;
     s := a * b - a div b;
```

НЕПРАВИЛЬНО:

```
  Var a, b: integer;
     r: integer;
        …
    r := a / b;
                    {деление по наклонной черте дает вещественный результат}

 Var a, b: real;
     r: integer;
        …
     r := a div b;        {операции div и mod используются
                          только для целочисленных данных}
```

1.7.2. Выражения и операции отношения

Выражением отношения называется словосочетание языка, в котором два выражения связаны знаком операции отношения. ***Операции отношения*** выполняют сравнение двух операндов и определяют, истинно значение выражения или ложно.

Сравниваемые величины могут принадлежать любому скалярному или перечисляемому типу данных. Результат всегда имеет ***булевский*** тип.

= – равно	<> – неравно
< – меньше	> – больше
<= – меньше или равно	>= – больше или равно

1.7.3. Логические операции и выражения

Логические выражения (ЛВ) используются преимущественно в операторах цикла и в условных операторах, когда от значения ЛВ зависит дальнейшее протекание процесса вычислений.

Обычно ЛВ строятся на основе арифметических выражений (АВ). Примером является отношение – запись равенства или неравенства, которая строится по схеме:

АВ <операция отношения> АВ.

Примеры отношений: `x >= 5`; `sqr(a) + sqr(b) <> sqr(c)`.

Значением ЛВ является «истина» (`True`) или «ложь» (`False`). Например, если текущее значение равно 6, то значение отношения `x >= 5` равно `True`.

Сложные ЛВ строятся из элементарных (например, отношений) с помощью логических (*булевских*) операций. Результатом выполнения логических операций является логическое значение `True` или `False`. Операндами служат данные типа `boolean`. Рассмотрим 3 из имеющихся 4 операций.

Если нужно проверить одновременную истинность нескольких ЛВ (например, отношений), их связывают операцией **`and`** (**конъюнкция**, логическое умножение). Конъюнкция имеет значение «ложь» (`False`) при ложности хотя бы одного из образующих ее логических компонентов.

🗎 **Рассмотрим пример.**

Нужно проверить, лежит ли точка (x, y) в секторе (0–45°) круга радиусом R, центр которого в начале координат.

Должны выполняться следующие условия:

– точка находится в круге, т.е. `sqr(x) + sqr(y) <= sqr(R)`;

– ее абсцисса неотрицательна, т.е. `x >= 0`;

– ее ордината не больше абсциссы, т.е. `y <= x`.

Получаем следующую запись логического выражения:

`(sqr(x) + sqr(y) <= sqr(R))` **`and`** `(x >= 0)` **`and`** `(y <= x)`.

Если для истинности всего условия достаточно истинности хотя бы одного из компонентов, то применяют операцию **`or`** (**дизъюнкция**, логическое сложение).

🗎 **Рассмотрим пример.**

Надо выделить значения переменной *x*, которые принадлежат либо интервалу (a_1, b_1), либо интервалу (a_2, b_2), либо интервалу (a_3, b_3).

Сразу заметим, что запись $a_1 < x < b_1$ отношением не является и воспринимается как ошибочная. Соответствующий ей смысл передает конъюнкция:

$$(a_1 < x) \text{ and } (x < b_1).$$

Для указанных значений переменной X истинно логическое выражение:

$$(a_1 < x) \text{ and } (x < b_1) \text{ or } (a_2 < x) \text{ and } (x < b_2) \text{ or } (a_3 < x) \text{ and } (x < b_3).$$

На языке Pascal это будет записано так:

`(a1 < x)` **`and`** `(x < b1)` **`or`** `(a2 < x)` **`and`** `(x < b2)` **`or`** `(a3 < x)` **`and`** `(x < b3)`.

Операция отрицания **`not`** (**инверсия**) изменяет смысл ЛВ на противоположный.

Пример:

$$\text{not } ((a_1 < x) \text{ and } (x < b_1) \text{ or } (a_2 < x) \text{ and } (x < b_2) \text{ or } (a_3 < x) \text{ and } (x < b_3)).$$

На языке Pascal:

`not` `((a1 < x)` **`and`** `(x < b1)` **`or`** `(a2 < x)` **`and`** `(x < b2)` **`or`** `(a3 < x)` **`and`** `(x < b3))`.

Смысл: «значение *x* не принадлежит ни одному из трех заданных интервалов».

Логические операции **`not`**, **`and`**, **`or`** и **`xor`** описываются в следующей таблице.

Операнд 1	Операнд 2	Результат для …			
		`not`	**`and`**	**`or`**	**`xor`**
`False`	–	`True`	–	–	–
`True`	–	`False`	–	–	–
`False`	`False`	–	`False`	`False`	`False`
`False`	`True`	–	`False`	`True`	`True`
`True`	`False`	–	`False`	`True`	`True`
`True`	`True`	–	`True`	`True`	`False`

1.7.4. Приоритет операций

Приоритетом называется очередность выполнения операций в выражении. Выполнение каждой операции проходит с учетом ее приоритета.

1. `not`, отрицание знака (унарный минус)
2. `and`, `shr`, `shl`, *, /, `div`, `mod`
3. +, -, `or`, `xor`
4. =, <>, <, >, <=, >=, `in`

Операции одного приоритета выполняются последовательно слева направо. Порядок выполнения операций можно изменить с помощью круглых скобок. Выражение, заключенное в скобки, трактуется как один операнд. Скобки могут быть вложены друг в друга:

```
a/(b*(c-b)).
```

При большом количестве скобок наглядность теряется, поэтому расчленяйте сложные арифметические выражения на более простые, то есть используйте больше присваиваний. Это тем более следует делать, если повторяются подвыражения или функция одного и того же аргумента. Вычисление подвыражений в отдельных операторах к тому же упрощает проверку работы программы.

1.7.5. Типы данных и допустимые операции

Для данных каждого типа допустимы свои операции.

Так, данные символьного типа допускают только операции сравнения (по кодам), данные логического типа допускают только логические операции. Результатом выполнения операций отношения (независимо от типа аргумента) и логических операций всегда будут данные логического типа.

Для данных числового типа операции сложения, вычитания и умножения одинаковы и для вещественных, и для целых чисел, а вот операции деления у них различны.

Почти все операции бинарны (то есть имеют два операнда, на схеме этоотображено более тонкими стрелками), кроме операции `not` – она унарная.

1.7.6. Основные математические функции

Для облегчения работы программиста наиболее распространенные алгоритмы вычисления математических функций (извлечение квадратного корня, значения тригонометрических функций и т.д.) разработаны фирмой-изготовителем и поставляются вместе с языком. Они хранятся в специальной библиотеке и называются встроенными функциями. Для того чтобы их использовать, необходимо знать синтаксис и правила обращения к ним. В таблице приведены некоторые из них.

Функция	Запись на языке Pascal	Тип аргумента	Тип результата
$\lvert x \rvert$	`abs` (x)	целый вещественный	целый вещественный
x^2	`sqr` (x)	целый вещественный	целый вещественный
$\sqrt{x}$	`sqrt` (x)	целый вещественный	вещественный
$\sin x$	`sin` (x)*	целый вещественный	вещественный
$\cos x$	`cos` (x)*	целый вещественный	вещественный
$\mathrm{arctg}\, x$	`arctan` (x)	целый вещественный	вещественный

e^x	`exp` (x)	целый вещественный	вещественный
$\ln x$	`ln` (x)	целый вещественный	вещественный
выделение дробной части аргумента	`Frac` (x)	вещественный	вещественный
выделение целой части аргумента	`Int` (x)	вещественный	вещественный
проверка четности	`Odd` (x)	целый	`boolean` (`true`, если число нечетное и `false`, если четное)
формирование случайного числа	`random` (x)	целый (`Word`)	целый (от 0 до x-1)
	`random`;	–	вещественный (от 0 до 1)
округление вещественного числа до целого	`Round` (x)	вещественный	целый
выделение целой части вещественного числа	`Trunc` (x)	вещественный	целый

* – аргумент функции должен быть в радианах.

1.7.7. Правила записи выражений в Pascal

При использовании алгебраических выражений помните, что их запись в тексте программ несколько отличается от математической записи.

1. Выражения, которые традиционно записываются с выходом из строки (например, $\frac{a+b}{c-d}$), в программе записываются в одну строку – $(a+b)/(c-d)$.
2. Знак операции умножения * нельзя опускать или заменять точкой.
3. Нельзя размещать два знака операций рядом. Последовательности символов 3*-2, $a/-b$ не являются выражением, необходимо записать: 3*(-2), $a/(-b)$.
4. При записи арифметических выражений с применением скобок необходимо соблюдать правило парности скобок, то есть число открывающихся скобок должно совпадать с числом закрывающихся скобок.
5. Пробелы обязательны при записи операций `div` и `mod`: *a* `div` *b*; *a* `mod` *b*. В отсутствие пробелов *adivb* и *amodb* будут интерпретироваться компилятором как идентификаторы.

1.7.8. Профилактика ошибок в арифметических выражениях

Одни ошибки выявляются автоматически на этапе компиляции (пропуск скобок, ошибочный тип аргумента или операнда и т.п.), другие – при вычислении арифметического выражения. К последним относятся деление на ноль, ошибочные значения аргументов математических функций, например, отрицательный аргумент функции `sqrt`.

Рассмотрим неожиданные и зачастую скрытые ошибки, связанные с применением типов. Напоминаем, что тип результата операции и диапазон значений определяется типом операндов.

🗎 **Пример**. Следующая программа:

```
Var  a: integer;
Begin
  a := 25000;
  writeln (a * a)
End.
```

выводит «нелепый» результат: –16832 вместо 625 000 000. Дело в том, что умножение производится в узком диапазоне значений типа `Integer`. Тип переменной `a` должен быть заменен на `Longint`.

Заметим, что следующий вариант программы также ошибочен:

```
Var  a: integer;
     b: longint;
Begin
  a := 25000;
  b := a * a;
  writeln (b)
End.
```

Дело в том, что тип переменной `b` никак не влияет на процесс вычисления выражения `a * a`, переполнение происходит и в `b` попадает уже испорченный результат умножения. Здесь не улучшит дела и такое «новшество»: `b := Longint(a * a)`, где воздействие также запоздало.

Будьте «морально готовы» к непредсказуемым проявлениям ошибок переполнения.

🗎 **Пример**. Случай «неожиданного» деления на ноль (ошибка 200):

```
Var  a: word;
     b: byte;
Begin
  a := 512;
  b := 128;
  writeln (1/(a * b))
End.
```

Значения `a` и `b` таковы, что их произведение лишь на 1 больше максимального числа в типе `Word` и переполнение дает 0 в знаменателе.

🗎 **Пример**. Ошибка, вызванная выходом из диапазона типа `Byte`.

```
Var  a, b: byte;
Begin
  a := 125;
  b := a * a;
  writeln (b)
End.
```

Вместо ожидаемого значения 15625 выводится «непонятное» число 9, но если выражение `a * a` выводить непосредственно, мы видим 15625. Что же происходит? До момента присваивания все идет нормально, но тип переменной `b` таков, что она не может «вместить» значение, которое больше 255. Происходит усечение значения. Переменной `b` надо задать тип `Word`.

Вплоть до завершения исчерпывающей проверки программы используйте контроль выхода из диапазона, включаемый записью директивы {$R+}.

Тогда присваивания переменным значений, не соответствующих типу переменных, вызовут сообщения об ошибке 201 и останов программы, причем курсор будет находиться в строке с ошибочным оператором.

Директива {$R+} записывается до контролируемых операторов, хотя бы и в описательной части программы. Так как контроль увеличивает время выполнения, предусмотрена и директива {$R-}, выключающая этот контроль и записываемая ниже всех «подозреваемых» операторов.

Итак, примите к исполнению следующие рекомендации:

1. *Создание и совершенствование программы осуществляйте поэтапно. Например, в начале используйте лишь типы* `Real`, `Integer` *из простых типов, добивайтесь безупречной работы программы, а затем осторожно проводите рационализацию в целях экономии памяти.*
2. *Избегайте многообразия типов простых переменных: экономия памяти может иметь «побочный эффект» смешения типов – увеличение числа ошибок.*
3. *Если в выражении, где на ваш взгляд возможно переполнение целых, есть деление (/), постарайтесь перестроить выражение так, чтобы деление выполнялось как можно ранее, ибо с этого момента работа пойдет в вещественном типе, диапазон которого широкий.*
4. *Не делайте сложных выражений; лучше иметь большее число простых операторов, размещаемых в отдельных строках, – проще локализовать ошибки.*

🕮 Вопросы для повторения:

1. Чем отличается операция деления / от операции `div`?
2. Существует ли в Pascal операция возведения в степень?
3. Может ли слово `Apple` являться переменной?
4. Что такое переменная?
5. Что такое идентификатор?
6. Какие из предложенных ниже записей являются идентификаторами, а какие нет?

   1) `ид` 2) `w9` 3) `9w` 4) `y(17)`
   5) `alpha` 6) `y#kl` 7) `_kl` 8) `f_h`
   9) `my prog` 10) `div` 11) `date_27` 12) `MyNameSerg`
7. Найдите и объясните ошибки в описаниях переменных целого типа:

```
Var a, b, c: integer;
   d, e, f, : integer;
   const: integer;
   d+t: integer;
   x,, y: integer;
   q, w, e, r, t: integer
```

8. Вычислите устно:

   1) `17 div 6` 2) `34 mod 8` 3) `19 div 4`
   4) `89 div 9` 5) `89 mod 9` 6) `77 mod 7`
9. Какие числа можно получить при вычислении выражения `x mod 5`?
10. Вычислите значения выражений:

    1) `trunk (2.8)` 5) `round (2.8)`
    2) `trunk (2.1)` 6) `round (2.1)`
    3) `trunk (-1.6)` 7) `round (-1.6)`
    4) `trunc (-1.1)` 8) `round (-1.1)`
12. Чему равен результат работы программ при a = 253:

```
Var a, x, y, z, b: integer;
Begin
   readln (a);
   x := a div 100;
   y := (a div 10) mod 10;
   z := a mod 10;
   b := x * 100 + z;
   writeln (b);
End.
```

13. Найдите и исправьте ошибки в программе:

```
Var  a, b: real;
     c, e: integer;
Begin
    writeln ('Введите два целых числа');
    readln (a);
    c := a / b;
    d := a mod b;
    e := c + d;
    writeln (c); writeln (d); writeln (e);
End.
```

Задания для самостоятельной работы:

1. Переменные A и B, типа Word, имеют значения 30153 и 35222 соответственно. Запишите и испытайте программу вычисления величины A^7/B^5. Должно получиться значение $4.18065 \cdot 10^8$ (округлено нами). Учтите рекомендацию 3
2. Составьте программы для вычисления следующих формул:

   а) $z = 3 + x^2 - 3x^3$ б) $d = x(\sin x + \cos x^2 + |x|)$

   в) $l = 2\pi r$ г) $k = 1 + |y - x|^3 + \dfrac{|y - x|}{x + 1}$

Глава 2. Операторы языка Pascal

Операторы языка Pascal можно разделить на простые и сложные. ***Простые*** не содержат внутри себя других операторов. ***Сложные (структурные)*** операторы представляют собой конструкции, содержащие протые операторы. К простым относятся следующие операторы: присваивания, операторы ввода и вывода. К сложным операторам относятся: составной оператор, оператор условного перехода, оператор выбора, операторы цикла.

2.1. Ввод и вывод данных

Для ввода и вывода данных в языке Pascal предусмотрены следующие процедуры ввода/вывода: `read`, `readln`, `write`, `writeln`. *Часто эти процедуры называют операторами по аналогии с другими операторами языка, хотя, строго говоря, называть процедуры операторами неверно.*

Для **ввода** информации с клавиатуры в компьютер используется процедура:

`Read` (<список переменных>);

В скобках необходимо записывать последовательность имен переменных, разделенных запятыми, которые мы хотим вводить при работе программы. При выполнении процедуры `read` программа останавливается и ждет, пока нужное количество чисел не будет введено с клавиатуры. Вводимые числа разделяют пробелами или нажатием клавиши «Enter». Заканчивают ввод всегда клавишей «Enter».

Процедура `readln` (...) отличается от `read` (…) тем, что, введя необходимое количество данных, пропускает все остальные, набранные до нажатия клавиши «Enter».

Отметим, что слово `read` *переводится с английского как «читать», а название оператора* `readln` *является сокращением от Read Line New (читать новая строка – дословный перевод).*

🗎 **Пример**. Рассмотрим следующую программу:

```
Var a, d: LongInt;
    b, c: Word;
    e: Integer;
begin
   Read(a, b);
   ReadLn(c, d);
   Read(e);
end.
```

Предположим, что пользователь ввел с клавиатуры следующие символы:

```
                10000
       20000        30000
      40000 abcbdba
50000
```

Вопрос: какие значения будут помещены в каждую из переменных в процессе выполнения программы?

Правильный ответ: оператор `Read` поместит в переменные `a` и `b` числа 10000 и 20000. Оператор `ReadLn` поместит в переменные `c` и `d` числа 30000 и 40000, остальные символы до символа перевода строки (то есть символы `'abcbdba'`) будут пропущены. Оператор `Read` попытается поместить в переменную `e` число 50000, но не сможет этого сделать, так как переменная имеет тип `Integer`, поэтому возникнет сообщение об ошибке.

Заметьте, что ошибка возникла не из-за того, что пользователь ввел посторонние символы `'abcbdba'`, а из-за того, что очередное введенное число не помещалось в диапазон данных для этого типа переменной, куда это число должно было быть помещено. Если бы в программе вместо оператора `ReadLn` был оператор `Read`,

то ошибка бы тоже возникла, но уже из-за попытки поместить в переменную типа `Integer` символов `'abcbdba'`.

Задание для самостоятельной работы.

На клавиатуре набраны две строки:

2 3 4 5
6

Какие значения придадут переменным X и Y процедуры:

а) `read` (X); `read` (Y);
б) `readln` (X); `read` (Y);?

Процедура `readln;` (без списка переменных) обычно записывается в конце программы. Она останавливает работу программы до нажатия клавиши «Enter». В противном случае по окончании работы программы окно текстового редактора закроет экран с полученными результатами.

Примечание.

Действие процедуры зависит от типа переменных. Если тип переменных:

- целый или вещественный, то *знаки пробелов и перевода строки перед числом игнорируются*;
- `Char`, то считывается *каждый* символ;
- `String`, то считывается максимум *q* символов при длине *q* строковой переменной.

Для **вывода** информации на экран компьютера используется процедура:

`Write` (<список параметров>);

В скобках необходимо записывать данные, которые будут выведены на экран при выполнении программы. Ими могут быть:

1) *Переменная.* В этом случае на экран выводится ее значение.
2) *Выражение.* В этом случае значение выражения вначале вычисляются, а затем выводится на экран.
3) *Константа.* Чтобы прокомментировать выводимые значения, в список вывода можно помещать строки любых символов, заключенные в *апострофы* (строковые константы).

Например:

```
Write ('Ответ: ', X, ' км/сек');
```

Эти строки появятся на экране без апострофов. Так при X = 5 предыдущая процедура выведет:

Ответ: 5 км/сек

Процедура `writeln` ( ) отличается от `write` ( ) тем, что после вывода всех значений переводит курсор в начало следующей строки. Можно использовать процедуру `writeln;` без параметров. Она только перемещает курсор в начало следующей строки.

2.2. Оператор присваивания

Оператор присваивания (**:=**) предписывает выполнить ***выражение***, заданное в его правой части, и присвоить результат ***переменной***, идентификатор которой расположен в левой части. Переменная и выражение должны быть совместимы по типу.

Например:

```
Y := 5;
X := sqr(Y) + 3;
```

Это означает следующее: переменной Y присвоить значение 5, а переменной X присвоить значение выражения 5^2+3, т.е. 28.

Таким образом, присваивание заполняет участок памяти, отведенный для переменной, новым значением, одновременно уничтожая старое. Поскольку задача любой программы – получить в определенном месте памяти нужное значение, редкая программа обходится без оператора присваивания.

Чтобы лучше понять работу оператора присваивания рассмотрим пример `a:= a+1`.

При выполнении этого оператора сначала будет вычислено значение выражения `a+1`, при этом вместо `a` в это выражение будет подставлено текущее значение `a`, затем значение этого выражения будет помещено в переменную `a`. Таким образом, после выполнения этого оператора присваивания переменная `a` увеличит свое значение на 1.

Если вещественной переменной присваивается целое значение, его представление изменяется автоматически. Присваивание вещественного значения переменной целого типа запрещено. Тип не может быть переопределен в программе, поэтому будьте внимательны при выборе типа и использовании переменных.

🕮 Вопросы для повторения:

1. Какая команда служит для ввода данных? Для вывода данных?
2. Чем отличается вывод информации на экран компьютера оператором `write (...)` от вывода оператором `writeln (...)`?
3. Как вывести на экран текст?
4. Можно ли вывести на экран несколько переменных или текстов одним оператором `writeln (...)`?
5. Что должно быть записано слева от оператора присваивания?
6. Что может быть записано в правой части оператора присваивания?

💻 Задания для самостоятельной работы:

1. Загрузите Pascal. Введите записанную ниже программу в компьютер. Просмотрите, какие разделы присутствуют в программе. Выполните программу.

```
const
   s = 'Моя первая программа';
Begin
   write (s);
End.
```

2. Введите записанную ниже программу в компьютер. На основании этого примера (удаляя и дописывая `ln`) попробуйте разобраться, в чём отличие между операторами `write ( )` и `writeln ( )`.

```
Var a: integer;
    b, c: real;
Begin
  a := -64;
  b := sqrt(abs(a));
  c := int(pi);
  writeln ('b = ', b);
  writeln ('c = ', c);
End.
```

3. Имеются два числа. Вычислите сумму и разность этих чисел, используя для ввода данных:
   a) два оператора `readln`;
   b) один оператор `readln`.
4. Имеются три числа. Вычислите их произведение.
5. По заданным сторонам прямоугольника a и b вычислите его периметр и площадь.
6. Человеку сегодня исполнилось R лет. Вычилите его возраст в днях (часах, минутах, секундах). (Високосные годы не учитывать)

2.2.1. Программирование линейных алгоритмов. Форматы вывода.

🗎 **Рассмотрим несколько примеров:**

Составим программу, вычисляющую $s = a + b$ *и* $p = a - b$.

```
Var a, b, s, p: real;
Begin
   readln (a);
   readln (b);
   s := a + b;
   p := a - b;
   writeln (s);
   writeln (p);
End.
```

В данном примере ввод и вывод данных для каждой переменной осуществляется отдельно. А можно было записать и так:

```
Var a, b, s: real;
Begin
   readln (a, b);
   s := a + b;
   p := a - b;
   writeln (s, ' ', p);
End.
```

В этой задаче использовался вещественный тип числа – `real`, поэтому ответ был получен в **экспоненциальной форме записи числа**.

Вспоминаем:

2.5670000000E+02 следует понимать $2.567 * 10^2$

3.4906710000E-03 следует понимать $3.490671 * 10^{-3}$.

Если мы хотим при выводе на экран реальных чисел указать определённое число знаков после запятой, то можно воспользоваться ***форматом вывода***

`writeln (x:p:q);`

Где **p** – количество знакомест, выделенных под всё число, в том числе и точку; **q** – количество знакомест, выделенных под дробную часть числа. Выводимый текст выравнивается по правому краю поля вывода.

Обратите внимание – если **p,** заданное программистом, мало, то при выводе ширина поля будет увеличена.

Например, команда `writeln (pi)` выведет строку:

3.1415926536E+00

а команда `writeln (pi:9:3)` выведет строку:

3.142

пропустив перед выводимым числом четыре пробела и выровняв 3.142 (всего 5 знаков) по правому краю девяти знакомест.

Составим программу, выводящую на экран число π.

```
Begin
   writeln (pi:9:3);
End.
```

Изменяя числа в формате вывода (:9:3, :9:2, :17:3, :17:7, :3:7 и т.д.), просмотрите и проанализируйте полученные результаты.

Задана сторона куба. Вычислить объем куба и площадь боковой поверхности с точностью 3 знака после запятой.

```
Var a, v, s: real;          {описание переменных}
Begin
  read (a);                 {ввод данных}
  v := a * a * a;           {вычисление объёма}
  s := 6 * a * a;           {вычисление площади поверхности}
  writeln (v:8:3, ' ', s:8:3);   {вывод результатов}
End.
```

Вопросы для повторения:

1. Что означает число 7 в команде `writeln (pi:7:5)`? А что означает число 5?
2. Сколько знаков после запятой будет выведено при записи команды:
   a) `writeln (pi:7:5);` b) `writeln (pi:5:7)`?
3. Какие символы используются для размещения комментариев в программе?

Задания для самостоятельной работы:

1. После начала некоторого эксперимента прошло t часов m минут и k секунд. Сколько всего секунд длится эксперимент?
2. Задано расстояние между городами в вёрстах. Переведите это расстояние в километры (точность: три знака после запятой) используя данные:
   1 верста = 500 саженям; 1 сажень = 3 аршина; 1 аршин = 0.7112 метра

2.2.2. Составление линейных алгоритмов с использованием арифметических операций

Рассмотрим несколько примеров:

Заданы два натуральных числа. Найдите их сумму.

```
Var a, b, s: byte;
Begin
   read (a, b);
   s := a + b;
   writeln (s);
End.
```

Проанализируйте полученный результат при `a + b > 255`. Поменяйте тип данных `byte` на `integer`. Что получается теперь? Просмотрите, до какой предельной суммы `a + b` вычисления происходят верно? Установите тип данных `longint`. Просмотрите суммы.

Заданы два целых числа k и d. Используя только арифметические операции, найдите целую и дробную части от деления k на d.

```
Var a, drobn: real;
      k, d, cel: integer;
Begin
   readln (k, d);
   cel := k div d;        {целочисленное деление}
   a := k / d;            {обычное деление}
   drobn := a - cel;
   writeln (cel);
   writeln (drobn:6:3);
End.
```

Можно ли было в данной программе все переменные определить как `integer` или как `real`? Попробуйте и проанализируйте.

Дано трёхзначное число. Чему равны его цифры?

```
Var a, c1, c2, c3: integer;
Begin
   readln (a);
   c1 := a div 100;
   c2 := a div 10 mod 10;
   c3 := a mod 10;
   writeln (c1);
   writeln (c2);
   writeln (c3);
End.
```

Дано действительное число a. Используя только пять операций умножения, получить a^{15}.

```
Var a, b, c, d, s: longint;
Begin
   readln (a);
   b := a * a;                {вторая степень}
   c := b * b;                {четвертая степень}
   d := c * a;                {пятая степень}
   s := d * d * d;            {пятнадцатая степень}
   writeln (s);
End.
```

🕮 Вопросы для повторения:

1. Чем отличается операция `/` от операции `div`?
2. Есть ли в Pascal операция возведения в степень?
3. Выполняется ли равенство: `(264 mod 100) div 10 = (368 div 10) mod 10`.
4. Придумайте формулу для нахождения предпоследней цифры целого числа.
5. Можно ли производить операции `mod` и `div` над переменными, которые описаны как `real`?
6. Как должна быть описана переменная `fl`, если она вычисляется по формуле `fl := a / b`? А как можно описать переменные `a` и `b`?
7. Как может быть описана переменная `k`, если она вычисляется по формуле `k := a div b`? А как нужно описать переменные `a` и `b`?

Задания для самостоятельной работы:

1. Задано двузначное число. Чему равна последняя цифра числа?
2. В двузначное число вписать ноль в середину и получить трехзначное число.
3. Найдите сумму цифр заданного трёхзначного числа.
4. Найдите цифры заданного четырёхзначного числа.
5. Найдите цифры и сумму цифр пятизначного числа.
6. С начала суток прошло `k` минут. Определите сколько сейчас часов и минут.
7. Идёт `k`-ая секунда суток. Определите который сейчас час (в часах, минутах и секундах).
8. Поезд перевозит `s` тонн груза. Сколько автомобилей грузоподъемностью `b` тонн (`b < s`) он способен заменить?
9. Дано действительное число `a`. Не пользуясь никакими другими операциями кроме умножения, получить:
   a) a^{13} за пять операций;
   b) a^{21} за шесть операций;

2.2.3. Составление линейных алгоритмов с использованием основных функций

Рассмотрим несколько примеров:

Заданы a, x, y. Вычислите $S = |a| + \sqrt{x + y^2}$

```
Var a, x, y, s: real;
Begin
   writeln ('введите число x ');
   readln (x, y, a);
   s := abs(a) + sqrt(x + sqr(y));
   writeln (s);
End.
```

Задан радиус (целое число). Вычислить площадь круга и длину окружности.

```
Var  r: integer;
     l, s: ???; {подумайте, какой тип данных необходимо записать}
Begin
   readln (r);
   s := pi * sqr(r);
   l := 2 * pi * r;
   writeln (s,' ', l);
End.
```

Заменяя в программе функцию int() на функции trunc(), round(), frac() и правильно используя типы данных, заполните таблицу:

	-3.7	-3.2	-3	3.2	3.7	3
`Int(x)`						
`Trunc(x)`						
`Round(x)`						
`Frac(x)`						

```
Var  a, d: real;
Begin
   readln (a);
   d := int(a);
   writeln (d);
End.
```

Вопросы для повторения:

1. В чём отличие функции `int(x)` от функции `trunc(x)`?
2. Чему равно: `int(-7.7)`?
3. Чему равно: `round(-7.7)`?
4. Придумайте три способа возведения числа b в четвёртую степень.
5. Вычислите: `sqr(sqrt(7))`.
6. С использованием каких функций можно вычислить `tg x`.

Задания для самостоятельной работы:

1. Задано число x. Вычислите s, если:
$$s = \cos x - \sin x + |x - 7| + x^2 - 1$$
2. Задано вещественное число f. Вычислите разность целой и дробной части числа f.
3. Задано трёхзначное (четырёхзначное) число. Найдите цифры заданного числа, используя только функцию:
   `int();` `trunc();` `frac()`.
4. Задана сторона *a* равностороннего треугольника. Вычислите высоту треугольника и его площадь.

2.3. Условный оператор

Если в зависимости от условия задачи или результатов промежуточных вычислений должны выполняться различные операторы, то используется **условный оператор.** Условный оператор может быть записан в полной и неполной форме.

Полная форма условного оператора

if <*условие*> **then** <*оператор 1*>
else <*оператор 2*>;

Неполная форма условного оператора

if <*условие*> **then** <*оператор*>;

Здесь **if, then, else –** зарезервированные слова (если, то, иначе соответственно).

Оператор выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. Если значение выражения есть `true` (истина), то выполняется <*оператор 1* >, указанный после слова **then** (то). Если результат вычисления выражения в условии есть `false` (ложь), то выполняется <*оператор 2*>. В неполной форме оператора – если результат выражения `true`, выполняется <*оператор*>, если `false` – оператор, следующий сразу за оператором **if**. Операторы **if** могут быть вложенными друг в друга.

🗎 **Рассмотрим несколько примеров:**

Даны два числа. Найти большее из них.

```
Var max, x, y: real;
Begin
   readln (x, y);
   if  x > y  then  max := x
              else  max := y;
   writeln (max);
End.
```

Даны три числа. Найти максимальное.

```
Var max, x, y, z: real;
Begin
   readln (x, y, z);
   if  x > y  then  max := x
              else  max := y;
   if  z > max  then  max := z;
   writeln (max);
End.
```

Задано число a. Кратно ли оно 3?

```
Var a: longint;
Begin
   readln (a);
   if  a mod 3 = 0  then  writeln (a, ' кратно 3')
                    else  writeln (a, ' не кратно 3');
End.
```

Дано число a ($a \neq 0$). Дать характеристику числа:

– положительное, отрицательное;

– чётное, нечётное;

– целое, не целое.

```
Var  a: real;
   k, m, p: string;
Begin
   readln (a);
   if a < 0 then k := ' отрицательное'
            else k := ' положительное';
   if a mod 2 = 0 then p := ' чётное'
                  else p := ' нечётное';
   if a = int(a) then m := ' целое'
                 else m := ' не целое';
   writeln ('число ', a:6:3, k, ' ; ', m, ' ; ', p);
End.
```

Дано три числа x, y, z. Вычислить сумму только положительных чисел из трех данных.

```
Var  x, y, z, sum: real;
Begin
   readln (x, y, z);
   sum := 0;
   if x > 0 then sum := sum + x;
   if y > 0 then sum := sum + y;
   if z > 0 then sum := sum + z;
   if sum > 0 then writeln (sum)
              else writeln ('Все отрицательные');
End.
```

Вопросы для повторения:

1. Какие формы записи оператора `if` вам известны?
2. Объясните исполнение оператора `if`.
3. Для чего в задаче 3_5 переменная `sum` первоначально была обнулена?

Задания для самостоятельной работы:

1. Даны два числа. Найти меньшее из них.
2. Даны три числа. Найти минимальное число.
3. Задано натуральное число *a*. Является ли оно чётным? Решите задачу,
   a) используя операцию `mod`;
   b) используя функцию `int`.
4. Задано натуральное число *a*. Является ли оно кратным 7?
5. Задано натуральное число *a*. Заканчивается ли оно на 0?
6. Дано трехзначное число. Кратна ли сумма его цифр шести?
7. Дано целое число $a > 9$. Больше ли цифра десятков цифры единиц? Дано двузначное число. Является ли сумма его цифр
   a) однозначным числом;
   b) двузначным числом.
8. Задано двузначное число. Дайте характеристику числа по следующему плану:
   - сумма цифр числа;
   - число чётное (нечётное);
   - первая цифра равна (неравна) второй цифре.
9. Даны два вещественных числа. Уменьшить первое число в пять раз, если оно больше второго по абсолютной величине.
10. Заданы три числа *x*, *y*, *z*. Вычислить произведение только отрицательных чисел из трех данных.
11. Заданы четыре целых числа *x*, *y*, *z*, *w*. Вычислить сумму только четных чисел из четырёх заданных.
12. Дано три числа *x*, *y*, *z*. Вычислить количество чисел, больших 7.

2.3.1. Простые и составные условия

Условия, которые мы до сих пор использовали, являются простыми. Если после служебного слова **`if`** проверяется только одно условие (a `mod` $3 = 0$; $a > b$; $k = 1$), то такое условие будем называть **простым условием**.

Если после служебного слова **`if`** возникает необходимость проверить сразу несколько условий, то группу таких простых условий будем называть **составным условием**. При написании составных условий каждое из простых условий, входящих в составное, берётся в скобки. Можно использовать союзы **`and`** (и) или **`or`** (или).

Рассмотрим следующий пример:

Даны два числа a, b (a, b ≠ 0). Ответить, одного ли знака введённые числа.

Разберём данную задачу.

Здесь возможны четыре варианта входных данных, для каждого из которых запишем отдельное составное условие:

a	b	Ответ:
$a > 0$	$b > 0$	Одного знака
$a < 0$	$b < 0$	Одного знака
$a > 0$	$b < 0$	Разные знаки
$a < 0$	$b > 0$	Разные знаки

```
Var a, b: real;
Begin
    readln (a, b);
    if (a > 0) and (b > 0) then writeln ('числа одного знака');
    if (a < 0) and (b < 0) then writeln ('числа одного знака');
    if (a > 0) and (b < 0) then writeln ('числа разных знаков');
    if (a < 0) and (b > 0) then writeln ('числа разных знаков');
End.
```

Попробуем уяснить для себя, что хороший программист всегда внимательно обдумывает даже уже написанную и отлаженную программу. Ведь почти всегда можно применить правило: "Любую программу можно переписать так, что она станет более короткой, либо более красивой".

А можно ли сделать нашу программу более красивой или более короткой? Да, можно. Давайте объединим первое и второе составные условия.

```
Var a, b: real;
 Begin
    readln (a, b);
    if (a > 0) and (b > 0) or (a < 0) and (b < 0)
               then writeln ('числа одного знака')
               else writeln ('числа разных знаков');
 End.
```

Программа стала действительно короче. Но составное условие состоит в свою очередь из двух составных условий.

А можно ли и эту программу сделать более красивой или более короткой? Можно. Приведём новую версию проверки условия для данной программы:

```
  if a * b > 0 then writeln ('числа одного знака')
               else writeln ('числа разных знаков');
```

Замените условие и убедитесь на компьютере, что оно работает верно.

Дано натуральное число. Проверить, является ли данное число двузначным, у которого первая цифра равна последней.

```
Var a: integer;
Begin
   readln (a);
   if (a mod 10 = a div 10) and (a > 9) and (a <= 99)
             then writeln ('YES')
             else writeln ('NO');
End.
```

Вопросы для повторения:

1. Что такое составное условие?
2. В чём отличие составного условия от простого?
3. Какие союзы используются при написании составного условия?

Задания для самостоятельной работы:

1. Заданы два числа. Являются ли они оба большими 20?
2. Заданы два числа. Являются ли они оба чётными?
3. Заданы два числа. Является ли хотя бы одно число чётным?
4. Заданы два числа. Является ли хотя бы одно число положительным?
5. Дано двузначное число. Являются ли обе цифры числа чётными?
6. Дано трехзначное число. Входит ли в него цифра 4?
7. Проверить, является ли число трехзначным, у которого первая цифра равна последней.
8. Определить, является ли данное целое число N нечетным четырехзначным числом.
9. Вывести на экран номер четверти, которой принадлежит точка с координатами (x,y), при условии, что x и y отличны от 0.
10. Дано натуральное число N ($N < 100$), определяющее возраст человека в годах. Дать для этого числа наименование: "год", "года", "лет".

2.3.2. Вложенные условные операторы

В качестве оператора на одной или обеих ветвях может выступать любой оператор, в том числе и условный. Полученная конструкция называется вложенными условными операторами.

В записи вложенных условных операторов возникает неоднозначность типа:

If <условие1> **Then** <оператор1> **If** <условие2> **Then** <оператор2> **Else**<оператор3>**;**

Неясно, к какому оператору `If` относится ветвь `Else`. Такая неоднозначность разрешается по следующему правилу: `Else` относится к ближайшему оператору `If`, у которого еще отсутствует данная ветвь.

2.4. Составной оператор

Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга точкой с запятой, и ограниченную операторными скобками **begin** и **end;** .

begin
<*оператор1*>;
<*оператор2*>;
…
<*операторN*>
end;

Составной оператор воспринимается как единое целое и может находиться в любом месте программы, где синтаксис языка допускает наличие оператора.

Например, после служебного слова **then** (или **else**) должен выполняться только один оператор. Если необходимо, чтобы выполнялось несколько операторов, их нужно взять в операторные скобки.

Для того чтобы при большом количестве операторных скобок программа была легко читаемой, **end;** желательно записывать под **begin**.

> *Даны действительные числа a, b (a ≠ b). Меньшее из этих двух чисел заменить их суммой, а большее - их произведением.*

```
Var a, b, sa, sb: longint;
Begin
   readln (a, b);
   sa := a;  sb := b;
   if a > b then
             begin
              b := sa + sb;
              a := sa * sb;
             end
            else
             begin
              a := sa + sb;
              b := sa * sb;
             end;
   writeln (a, ' ', b);
End.
```

Составить программу, решающую квадратное уравнение вида $ax^2+bx+c=0$.

```
Var  a, b, c: integer;
   x, d, x1, x2: real;
Begin
  readln (a, b, c);
  d := b * b - 4 * a * c;  {вычисление дискриминанта}
  if d < 0 then   writeln ('уравнение  не  имеет  корней');
  if d = 0 then
    begin
      writeln ('уравнение  имеет  1  корень');
      x := -b / (2 * a);
      writeln ('x = ', x:6:3);
    end;
  if d > 0 then
    begin
      writeln ('уравнение  имеет  2 корня');
      x1 := (-b + sqrt(d)) / (2 * a);
      x2 := (-b - sqrt(d)) / (2 * a);
      writeln ('x1 = ', x1:6:3);
      writeln ('x2 = ', x2:6:3);
    end;
End.
```

Вопросы для повторения:

1. Что представляет собой составной оператор?
2. Для чего в задаче 3_8 введены переменные *sa*, *sb*?

Задания для самостоятельной работы:

1. Даны действительные числа x, y ($x \neq y$). Меньшее из этих двух чисел заменить их полусуммой, а большее - их удвоенным произведением.
2. Даны два целых числа M, N. Если M делится нацело на N, то вывести на экран частное от деления, в противном случае – сообщение "M на N нацело не делится".
3. Даны три числа. Удвоить их, если они упорядочены по возрастанию, иначе отрицательные заменить их модулями.
4. Даны три числа. Удвоить их, если они целые, иначе нецелые заменить их целыми частями.
5. Дано натуральное число n ($n \leq 9999$). Если число четырёхзначное, то получите и выведите перевёртыш этого числа (3528 → 8253), иначе выведите ответ "Число не четырехзначное".
6. Дано натуральное число n. Если оно двузначное, то впишите в середину ноль (для 56 → 506), иначе выведите ответ "Число не двузначное".
7. * Даны натуральные числа k, l ($1 \leq k, l \leq 8$). На клетке (k, l) расположена шахматная фигура:

   a) конь b) ферзь

   Вывести на экран координаты клеток, на которые может попасть за один ход данная фигура, при этом учитывайте, что нельзя выходить за пределы доски.

2.5. Оператор выбора

Если один оператор **if** может обеспечить выбор из двух альтернатив, то оператор выбора **case** позволяет выбрать из *нескольких* возможных вариантов.

🗎 **Пример 1.**

Предположим, что нам нужно ввести с экрана число A (0 ≤ A ≤ 9) и напечатать его словами (например, если A = 1, то напечатать «один»). Понятно, что эту задачу несложно решить с помощью использования оператора if. Программа приведена ниже:

```
Var A: integer;
begin
  ReadLn(A);
  if A = 0 then WriteLn('ноль');
  if A = 1 then WriteLn('один');
  if A = 2 then WriteLn('два');
  if A = 3 then WriteLn('три');
  if A = 4 then WriteLn('четыре');
  if A = 5 then WriteLn('пять');
  if A = 6 then WriteLn('шесть');
  if A = 7 then WriteLn('семь');
  if A = 8 then WriteLn('восемь');
  if A = 9 then WriteLn('девять');
end.
```

Однако программа очень сильно нагружена операторами if. Оказывается, вместо десяти операторов if можно использовать всего один оператор выбора **case**.

Структура оператора выбора такова:

case <*выражение*> **of**
<*список констант*>: <*оператор1*>;
<*список констант*>: <*оператор2*>;
...
<*список констант*>: <*операторN*>;
[**else** <*операторы*>]
end;

Оператор **case** работает следующим образом. Вычисляется значение выражения, его значение отыскивается в одном из списков констант. После этого выполняется оператор, соответствующий списку.

Решение примет следующий вид:

```
Var A: integer;
begin
  ReadLn(A);
  case A of
    0: WriteLn('ноль');
    1: WriteLn('один');
    2: WriteLn('два');
    3: WriteLn('три');
    4: WriteLn('четыре');
    5: WriteLn('пять');
    6: WriteLn('шесть');
    7: WriteLn('семь');
    8: WriteLn('восемь');
    9: WriteLn('девять')
  end;
end.
```

Если значение выражения не найдено в списках, не выполняется ни один оператор (или выполняются операторы, стоящие после `else`).

🗎 **Пример 2.**

Написать программу, которая вводит число `A` (`1 ≤ A ≤ 9`) и выводит, является оно четным или нечетным. Решение имеет следующий вид:

```
Var A: integer;
begin
  ReadLn(A);
  case A of
    2, 4, 6, 8: WriteLn('четное')
    else WriteLn('нечетное')
  end;
end.
```

🗎 **Пример 3.**

Написать программу, которая вводит число `A` (`0 ≤ A ≤ 999`) и выводит, является оно однозначным, двузначным или трехзначным. Решение имеет следующий вид:

```
Var A: integer;
begin
  ReadLn(A);
  case A of
    0..9: WriteLn('однозначное');
    10..99: WriteLn('двузначное')
    else WriteLn('трехзначное')
  end;
end.
```

При использовании оператора выбора **`case`** должны выполняться следующие правила:

1. Значения выражения, записанного после служебного слова **`case`**, должны принадлежать порядковому типу. Чаще всего – `integer`, реже – `char`, `Boolean` или один из пользовательских типов. Использование вещественного и строкового типа недопустимо.
2. Тип констант списков должен совпадать с типом выражения.
3. все константы должны быть уникальны в пределах оператора **`case`** (то есть повторения констант в списках не допускаются); диапазоны не должны пересекаться.

Рассмотрим еще несколько примеров:

Вводится число от 1 до 4, определяющее время года. Дать название этого времени года (1 – зима, 2 – весна, 3 – лето, 4 – осень).

```
Var n: byte;
Begin
  readln (n);
  case n of
   1: writeln ('зима');
   2: writeln ('весна');
   3: writeln ('лето');
   4: writeln ('осень');
  else
   writeln ('неправильно введен номер времени года');
  end;
End.
```

*Составим программу "КАЛЬКУЛЯТОР", которая после ввода двух чисел и одного из знаков +, -, *, / произведёт вычисления, а результат выдаст на экран.*

```
Var a, b, s: real;
     sim: char;
Begin
   writeln ('Введите два числа');
   readln (a, b);
   writeln ('Введите знак операции');
   readln (sim);
   case sim of
      '+': s := a + b;
      '-': s := a - b;
      '*': s := a * b;
      '/': s := a / b;
   end;
   writeln (s);
End.
```

В данной программе отсутствует часть `else`, поэтому, если ввести вместо рассматриваемых арифметических знаков, ввести любой символ, то программа будет работать, но будет работать неверно.

Вопросы для повторения:

1. Сколько строк может быть записано в списке выбора?
2. Может ли в операторе выбора отсутствовать часть **`else`**?
3. Сформулируйте, что может являться ключом выбора.
4. Можно ли оператор выбора заменить условным оператором **`if ... then`**?
5. Сколько операторов **`if ... then`** понадобилось бы для решения задачи 3_10?

Задания для самостоятельной работы:

1. Отредактируйте последнюю задачу так, чтобы при вводе произвольного символа программа выдавала ответ: "Введена некорректная арифметическая операция"; Вводится число от 1 до 10. Дать название этого числа (1 – один, 2 – два, …, 10 – десять);
2. Вводится число от 1 до 7, определяющее день недели. Дать название этого дня (1 – понедельник, 2 – вторник, …, 7 – воскресенье);
3. В спортивных соревнованиях Шарик, кот Матроскин, дядя Фёдор и почтальон

Печкин заняли соответственно 1, 2, 3 и 4 места. Составить программу, которая по номеру места выдаёт имя участника соревнований.

4. Вводится число от 1 до 15. Вывести данное число, записанное римскими цифрами (I, II, III, IV, V, VI, …., XV);
5. Вводится число от 1 до 15. Вывести данное число, записанное в двоичной системе счисления (1, 10, 11, 100, 101, …., 1111);
6. Вводится число от 1 до 12, определяющее месяц года. Дать название этого месяца года (1 – Январь, 2 – Февраль, …, 12 – Декабрь);
7. Вводится номер месяца. Вывести время года для этого месяца (1 – зима, …, 3 – весна, …, 8 – лето, …);
8. Вводится номер времени года. Вывести названия месяцев для этого времени года (1 – декабрь, январь, февраль, …);
9. *Вводится число от 1 до 100. Дать название этого числа (1 – один, 2 – два, …, 100 – сто);
10. Дано натуральное число N ($N < 20$), определяющее сумму денег в рублях. Дать для этого числа наименование: "рубль", "рубля", "рублей";
11. Вводится число от 1 до 7, определяющее день недели. Вывести расписание уроков в вашем классе в этот день.

2.6. Операторы повтора (циклы)

Цикл – это замечательное изобретение, которое, в сущности, и делает компьютеры такими ценными. Он позволяет многократно повторить любую часть программы. Без такой возможности для одной секунды работы компьютера потребовалось бы писать тысячи строк программы.

В языке Pascal существует три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программы (три оператора цикла):

- цикл со счетчиком – цикл **FOR**;
- цикл с предусловием – цикл **WHILE**;
- цикл с постусловием – цикл **REPEAT**…**UNTIL** .

2.6.1. Цикл с параметром FOR

Формат записи цикла:

```
For  i := n  to  k  do <оператор>;
```

Переменная i (параметр или счетчик цикла) изменяется автоматически от n (начальное значение параметра) до k (конечное значение параметра, $k > n$) с шагом 1, поэтому она должна иметь порядковый тип.

Например:

```
For i := 1 to 10 do a[i] := 0;
For c := 'A' to 'Z' do writeln (c);
```

Вместо переменных n, k могут быть выражения того же типа.

Выполнение начинается с вычисления значений выражений начального и конечного параметра. Затем переменная i получает значение n и выполняется проверка, не превышает ли оно значение k. Если окажется, что $\boldsymbol{n > k}$ цикл не выполнится ни разу и управление передается следующему после `For` оператору. Иначе выполняются следующие действия: выполняется оператор тела цикла. После завершения оператора переменная получает следующее по порядку значение, и все повторяется, начиная с проверки.

Когда значение переменной n становится равным k, оператор выполняется последний раз.

Таким образом, при $\boldsymbol{n < k}$ *цикл выполнится* $\boldsymbol{(k - n + 1)}$ *раз, при* $\boldsymbol{n = k}$ *цикл выполнится только* ***один*** *раз, при* $\boldsymbol{n > k}$ *цикл не выполнится* ***ни разу****.*

Заметьте, что начальное и конечное значение перематра вычисляются ровно один раз в начале цикла.

Оператор цикла с параметром имеет вариант, когда переменная принимает последовательно убывающие значения:

```
For  i:= n  downto  k  do <оператор>;
```

В этом случае, чтобы цикл выполнился хотя бы один раз, начальное значение параметра должно быть больше его конечного значения.

Например:

```
For i := 10 downto 1 do a[i] := 0;
For c := 'Z' downto 'A' do writeln (c);
```

Если в цикле должны выполняться несколько операторов, то используем операторные скобки: **`begin … end;`**.

Цикл `FOR` удобно использовать тогда, когда точно известно количество повторений.

Примечание. *После* выполнения оператора `For` ***значение параметра*** цикла ***не определено***, поэтому запретим использование его значения после выполнения оператора `For`.

🗎 **Рассмотрим несколько примеров:**

> *Найти сумму всех натуральных чисел от 1 до n.*

```
Var i, n, s: integer;
Begin
   readln (n);
   s := 0;
   for  i := 1  to  n  do s := s + i;
   writeln (s);
End.
```

В данном цикле переменная *i* автоматически изменяется от 1 до *n* с шагом 1. Поэтому к переменной *s* прибавляется *i* вначале равная 1, потом 2, потом 3, и т.д. до *n*, соответственно переменная *s* принимает значения 1, 3, 6, 10, 15….

> *Определить количество трёхзначных натуральных чисел, сумма цифр которых равна заданному числу N.*

```
Var c1, c2, c3, i, n, kol: integer;
Begin
   readln (n);
   kol := 0;
   for  i := 100  to  999 do
     begin
        c1 := i div 100;
       c2 := (i div 10) mod 10;
        c3 := i mod 100;
        if  c1 + c2 + c3 = n  then  kol := kol + 1;
     end;
   writeln (kol);
End.
```

🕮 **Вопросы для повторения:**

1. Для чего предназначен оператор цикла?
2. Какие существуют циклы в языке Pascal?
3. Какой формат записи имеет оператор `FOR`?

4. Как работает оператор `FOR`?
5. Сколько раз выполнится цикл `FOR i := n downto k do...` при $n > k$? При $n = k$? При $n < k$?
6. В каких случаях применяется оператор `FOR`?
7. Сколько раз будет выполнен цикл, и чему будет равна переменная S после выполнения:

```
s := 0;  n := 6;
for   i := 3   to   n   do s := s + i;
```

8. Как в теле цикла выполнить несколько операторов?

Задания для самостоятельной работы:

1. Найти сумму всех нечётных трёхзначных чисел.
2. Найти сумму положительных чисел кратных 7, меньших 100.
3. Найти все числа, которые делятся на N, среди:
   a) всех двузначных чисел;
   b) всех трёхзначных чисел.
4. Составить программу вычисления суммы квадратов чисел от 1 до n.
5. Среди двузначных чисел найти те, сумма квадратов цифр которых делится на 13.
6. Дано натуральное число n. Вычислить факториал этого числа. *Примечание*: факториал числа n (обозначается $n!$) равен произведению всех натуральных чисел от 1 до n. То есть $n! = 1 * 2 * 3 * \ldots * n$
7. Дано натуральное число n. Вычислить:
   a) 2^n; b) 3^n;
8. Найти все делители для заданного натурального числа n.
9. Среди четырёхзначных чисел выбрать те, у которых:
   a) все четыре цифры различны (например: 3167, 9012);
   b) имеются три одинаковые цифры (например: 1311, 7779);
   c) цифры попарно различны (например: 1331, 7979, 2255);
   d) цифры образуюn возрастающую последовательность (например: 1389, 4678);
10. Написать программу поиска чисел < 1000, которые при делении на 2 дают в остатке 1, при делении на 3 дают в остатке 2, при делении на 4 – в остатке 3, при делении на 5 – в остатке 4, при делении на 6 – в остатке 5, а при делении на 7 дают в остатке 6.

2.6.2. Циклы с условиями

Цикл **`FOR`** используется, когда число повторений заранее известно. Если количество повторений тела цикла зависит от выполнения или невыполнения какого-либо условия, то применяются циклы с условиями **`While`** и **`Repeat..Until`**.

- цикл с предусловием:

`While` <*условие*> **`do`** <*оператор*>;

Работает этот оператор следующим образом. Вычисляется значение *логического выражения*, стоящего в условии. Если получается истина (`True`), то выполняется *оператор*, а затем снова вычисляется значение *логического выражения*. Если снова получается истина, то опять выполняется *оператор*, и т.д. Так продолжается до тех пор, пока при вычислении *логического выражения* не получится ложь (`False`). После этого оператор `While` заканчивает свою работу и передает действие следующему за ним оператору.

В частности, если в самом начале работы `While` при вычислении *логического выражения* получается ложь, то *оператор* не выполнится ни разу. Как обычно, в качестве *оператора* может выступать некоторый составной оператор.

Для того, чтобы оператор `While` смог закончить свою работу, переменные, от которых зависит значение *логического выражения,* должны измениться хотя бы один раз во время выполнения *оператора*, что повлечет за собой изменение значения *логического выражения*. Не исключена ситуация, когда *логическое выражение* всегда будет истинным, и оператор `While` будет работать бесконечное число раз. Такая ситуация называется *зацикливанием*.

Таким образом, при использовании оператора `While` и вообще других циклических операторов нужно стараться избегать *зацикливаний*. Это значит, что при программировании любого цикла нужно стараться всегда объяснить самому себе, почему этот цикл не будет бесконечным, а когда-нибудь закончит свою работу.

Самым простым примером вечного цикла может быть следующий:

While `True` **do** <*оператор*>;

- цикл с постусловием:

```
Repeat
   <оператор1>;
   <операто2р>;
   …
   <операторN>
Until <условие>;
```

Работает этот оператор достаточно похоже на `While`. Существенных отличий два:

1) Группа операторов в теле цикла выполняется хотя бы один раз (в самом начале цикла).

2) Цикл заканчивает свою работу, когда при вычислении логического выражения получается истина (`True`).

Опишем работу цикла `Repeat..Until` более точно. Сначала выполняются все операторы в теле цикла. Далее вычисляется логическое выражение, записанное в условии. Если оно ложно, то снова выполняются все операторы из цикла. Затем опять вычисляется логическое выражение и т.д. Цикл заканчивает свою работу, как только при вычислении логического выражения получается истина (`True`). Таким образом, если значение логического выражения в условии сразу ложно, то тело цикла выполнится ровно один раз.

Найти сумму всех натуральных чисел от 1 до n.

1) цикл FOR

```
Var i, n, s: integer;
Begin
     readln (n);
     s := 0;
     for  i := 1  to  n  do
                  s := s + i;
     writeln (s);
End.
```

2) цикл WHILE

```
Var i, n, s: integer;
Begin
   readln (n);
   s := 0;
   i := 1;
   while   i <= n   do
        begin
           s := s + i;
           i := i + 1;
        end;
   writeln (s);
End.
```

Цикл `WHILE` будет выполняться до тех пор, пока истинно логическое выражение *i* <= *n*. Причем переменную *i*, влияющую на значение логического выражения, обязательно изменяем внутри тела цикла. При невыполнении этого условия получаем пример того, что называется «зацикливанием» - бесконечный цикл.

Значение логического выражения вычисляется ***перед*** каждым повторением оператора. Если необходимо повторять несколько операторов, их следует объединить в составной оператор.

3) цикл REPEAT

```
Var i, n, s: integer;
Begin
    readln (n);
    s := 0;   i := 1;
    Repeat
          begin
               s := s + i;
               i := i + 1;
          end;
    Until   i > n;
    writeln (s);
End.
```

Цикл `Repeat ... Until` будет выполняться до тех пор, пока логическое выражение $i > n$, стоящее после слова `Until`, не станет истинным.

В отличие от оператора `While`, вычисление логического выражения происходит не до, а ***после*** очередного повторения цикла. Из-за этого цикл ***`Repeat` обязательно выполнится хотя бы один раз***, а цикл ***`While` может не выполниться ни разу***.

При использовании оператора `Repeat ... Until` необходимо учитывать следующее:

- тело цикла должно содержать хотя бы один оператор, влияющий на значение логического выражения, иначе получим бесконечный цикл;
- логическое выражение в конечном итоге должно принять значение `True`.

Задано натуральное число n. Вычислить сумму цифр этого числа.

```
Var n, sum, cif: integer;
Begin
   readln (n);
   sum := 0;
   while  n > 0  do
          begin
            cif := n mod 10;
            sum := sum + cif;
            n := n div 10;
          end;
   writeln (sum);
End.
```

Найти минимальное натуральное число, которое при делении на 2 даёт в остатке 1, при делении на 3 даёт в остатке 2, при делении на 4 - в остатке 3, при делении на 5 - в остатке 4, при делении на 6 - в остатке 5, а при делении на 7 дают в остатке 6.

```
Var i, kl: longint;
Begin
  kl := 0;
  i := 0;
  while   kl = 0   do
    begin
     i := i + 1;
     if (i mod 2 = 1) and (i mod 3 = 2) and (i mod 4 = 3) and
        (i mod 5 = 4) and (i mod 6 = 5) and (i mod 7 = 6)
             then kl := 1;
    end;
  writeln (i);
End.
```

Вопросы для повторения:

1. Какие циклы существуют в языке Pascal?
2. Какой формат записи имеют циклы `WHILE` и `REPEAT`?
3. В каких случаях удобно применять эти циклы?
4. Чем отличается цикл `WHILE` от цикла `REPEAT`?
5. Будет ли остановлено выполнение данного цикла? Почему?

```
s := 0;
i : = 1;
while   i <= 4   do   s := s + i;
```

Задания для самостоятельной работы:

1. Дано натуральное число *n*.
   a) Сколько цифр в числе *n*?
   b) Сколько чётных цифр в числе *n*?
2. Дано натуральное число *n*.
   a) Вычислить, входит ли цифра 3 в запись числа *n*.
   b) Поменять порядок цифр числа *n* на обратный.
   c) Переставить первую и последнюю цифры числа *n*.
   d) Приписать по единице в начало и в конец записи числа *n*.
   e) Является ли число *n* палиндромом? (9889 – да, 9878 – нет)
3. Дано натуральное число *n*. Является ли *n* степенью 3?
4. Для данного натурального числа $m > 1$ найдите максимальное k, для которого ещё выполняется равенство $2^k < m$. (например, если $m = 10$, то $k = 3$).
5. Для данного натурального числа $m > 1$ найдите минимальное k, для которого уже выполняется равенство $k! > m$. (например, если $m = 10$, то $k = 4$).

2.6.3. Вложенные циклы

Для решения задачи достаточно часто требуется использовать несколько вложенных друг в друга циклических конструкций, то есть оператор, повторяемый в цикле, сам может быть циклом. Такие конструкции называют ***вложенными циклами***.

Внешний и внутренний циклы могут быть любыми из трех рассмотренных ранее видов. При использовании вложенных циклов необходимо соблюдать следующее условие: внутренний цикл должен ***полностью*** укладываться в тело внешнего цикла.

В Pascal нет ограничений на количество и глубину вложения циклов.

Рассмотрим несколько примеров:

> *Дано натуральное число S. Требуется написать программу для нахождения всех прямоугольников, площадь которых равна S и стороны выражены натуральными числами.*

```
Var s, a, b: longint;
Begin
  readln (s);
  for  a := 1  to  s  do
    for  b := 1  to  s  do
      if  a * b = s  then  writeln (a, ' ', b);
End.
```

Данную задачу можно было решить, используя только один цикл. Подумайте, как это сделать.

> *Даны натуральные числа n, m. Получить все натуральные числа, меньшие n, сумма квадратов цифр которых равна m.*

```
Var  n, m, i, a, sum, cif: longint;
Begin
    readln (n, m);
    for  i := 1  to  n  do
       begin
          a := i;
          sum := 0;
          while  a > 0  do
              begin
                cif := a mod 10;
                sum := sum + sqr(cif);
                a := a div 10;
              end;
          if  sum = m  then  write (i, ' ');
      end;
End.
```

> *Найти все решения заданного числового ребуса. Каждой букве соответствует некоторая цифра. Причём одинаковым буквам соответствуют одинаковые цифры, разным буквам - разные цифры.*

$$\begin{array}{r} \text{КТО} \\ +\ \underline{\text{КОТ}} \\ \text{ТОК} \end{array}$$

Поскольку здесь всего три буквы, то для решения достаточно написать три вложенных цикла, чтобы перебрать все варианты сложения трёхзначных чисел.

```
Var k, t, o, kto, kot, tok: longint;
Begin
 for  k := 0  to  9  do
  for  t := 0  to  9  do
   for  o := 0  to  9  do
    begin
     kto := k * 100 + t * 10 + o;
     kot := k * 100 + o * 10 + t;
     tok := t * 100 + o * 10 + k;
     if (k <> t) and (k <> o) and (t <> o) and (kto + kot = tok)
                  then  writeln (kto, ' + ', kot, ' = ', tok);
     end;
End.
```

В данном алгоритме тело цикла выполнялось 10·10·10=1000 раз (будем говорить, что сложность алгоритма = 1000).

Если же для решения более сложных ребусов потребуется написать 8–10 вложенных циклов, то такой полный перебор будет работать достаточно долго.

Можно немного упростить данный алгоритм, если увидеть, что $1 \le k \le 4$, $t \ge 2$.

```
for  k := 1  to  4  do
   for  t := 2  to  9  do
      for  o := 0  to  9  do
```

Теперь сложность алгоритма 4·8·10=320. Простое косметическое исправление дало увеличение скорости в 3 раза.

Но и данный алгоритм не является оптимальным. Посмотрите, при $k = 2$ и $t = 2$ программа переберёт все 10 вариантов переменной o. В таких случаях, когда $k = t$, цикл по переменной o вообще необходимо не выполнять.

Назовём такой метод – контролируемый перебор.

```
Var k, t, o, kto, kot, tok: longint;
Begin
 for  k := 1  to  4  do
  for  t := 2  to  9  do
   if  k <> t then
    for  o := 0  to  9  do
     if  (k <> o) and (t <> o) then
      begin
       kto := k * 100 + t * 10 + o;
       kot := k * 100 + o * 10 + t;
       tok := t * 100 + o * 10 + k;
       if  kto + kot = tok then
          writeln (kto, ' + ', kot, ' = ', tok);
      end;
End.
```

Такой алгоритм работает очень быстро.

Вопросы для повторения:

1. Какие циклы называются вложенными?
2. Какому условию должны удовлетворять вложенные циклы?
3. Какие утверждения являются верными (проиллюстрируйте ответ соответствующими примерами):
   a. любой цикл `For` можно записать при помощи цикла `While`;
   b. любой цикл `While` можно записать при помощи цикла `For`;
   c. не любой цикл `For` можно записать при помощи цикла `While`;
   d. не любой цикл `While` можно записать при помощи цикла `For`;

Задания для самостоятельной работы:

1. *Старинная задача.* Сколько можно купить быков, коров и телят, если бык стоит 10 рублей, корова – 5 рублей, телёнок – полтинник (0,5 рубля), при условии, что на 100 рублей надо купить 100 голов скота.
2. Задано натуральное n. Для всех чисел от 1 до n найти:
   a) количество делителей; b) сумму чётных делителей.
3. Найти все решения следующих числовых ребусов:
   a) БАБКА + ДЕДКА + РЕПКА = СКАЗКА (4 решения)
   b) КОРОВА + ТРАВА + ДОЯРКА = МОЛОКО (2 решения)
   c) АЛЁНКА + ИВАН + КОЗЛИК = СКАЗКА (1 решение)
   d) ВЕТКА + ВЕТКА + СТВОЛ = ДЕРЕВО (3 решения)
   e) ВОРОТА + ТРАВА = ФУТБОЛ (3 решения)

Глава 3. Массивы

Потребность использовать массив возникает всякий раз, когда при решении задачи приходится иметь дело с большим, но конечным количеством однотипных данных, которые необходимо хранить в памяти.

Например, произведены наблюдения за температурой воздуха в течение некоторого периода времени (например, месяца). Закончив наблюдения, приступаем к обработке полученных данных: поиску самого холодного или теплого дня, вычислению средней температуры и т.д. Для этого мы должны составить алгоритм и программу, осуществляющие обработку данных, которые в подобном случае удобно хранить в массиве.

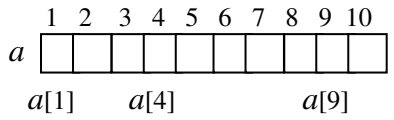
Массивы – это структурированный тип данных, состоящий из фиксированного числа элементов, которые имеют один и тот же тип. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Тип элементов массива называется ***базовым*** типом.

Для описания массива используется зарезервированное слово **`array`**.

Например, одномерный (линейный) массив, состоящий не более чем из 10 целых чисел, можно описать следующим образом:

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

Доступ к каждому элементу массива осуществляется путем их индексирования.



В данном примере мы взяли диапазон изменения индекса от 1 до 10. Можно было бы взять и диапазон от 0 до 9. Вообще, можно было в качестве границ взять любые целые числа, разность между которыми равна 9. Необходимо лишь, чтобы номер последнего элемента был больше, чем номер первого. Например, если в задаче данные распределены по годам, то в качестве индекса можно взять номер года – `a: array [1901..1910] of real`.

Элементы массива сами могут быть типа массив. Например, массив из 10-ти элементов, каждый из которых является массивом из 20-ти элементов типа `real`:

```
b: array [1..10] of array [1..20] of real;
```

Для краткости и удобства многомерные массивы можно описывать и более простым способом:

```
Var b: array [1..10, 1..20] of real;
```

Такой массив называют ***двумерным***. Подробнее работу с двумерным массивом рассмотрим в главе 3.3.

Если количество индексов равно *n*, то массив называют ***многомерным*** (*n*-мерным). Количество индексов массива часто называют его ***размерностью***.

Индексы представляют собой выражения любого порядкового типа (перечислимый, интервальный, символьный, логический, а также произвольный тип, созданный на их основе).

Правила употребления индексов при обращении к компонентам массива таковы:

1. Индекс компоненты может быть константой, переменной или выражением, куда входят операции и вызовы функций.
2. Тип каждого индекса должен быть совместим с типом, объявленным в описании массива именно для соответствующего «измерения»; менять индексы местами нельзя.
3. Количество индексов не должно превышать количество «измерений» массива. Попытка обратиться к линейному массиву как к многомерному обязательно вызовет ошибку. А вот обратная ситуация вполне возможна: например, если вы

описали N-мерный массив, то его можно воспринимать как линейный массив, состоящий из $(N-1)$-мерных массивов.

Элементы массива располагаются в памяти компьютера последовательно. Элементы с меньшим значением индекса хранятся в более низких адресах памяти. В многомерных массивах первым возрастает самый правый индекс.

Общий размер массива не должен превосходить 65 520 байт (64 Кбайт – размер памяти, отводимой под переменные программы). Следовательно, попытка задать массив

```
a: array[integer]of byte;
```

не увенчается успехом, поскольку тип `integer` покрывает 65 535 различных элементов.

Если ваша программа должна обрабатывать матрицы переменных размерностей (скажем, N по горизонтали и M по вертикали), то вы столкнетесь с проблемой изначального задания массива, ведь в разделе `Var` не допускается использование переменных. Следовательно, следующее описание массива

```
Var m, n: integer;
    a: array[1..m, 1..n] of real;
```

использовать нельзя.

Предположим, что вам известны максимальные границы индексов обрабатываемого массива. Скажем, N и M заведомо не могут превосходить 100. Тогда можно выделить место под наибольший возможный массив, а реально работать только с указанной пользователем его частью:

```
const  k =100;
Var a: array[1..k, 1..k] of real;
    m, n: integer;
```

В ряде случаев удобно применять массивы, описанные как ***типизированные константы*** в разделе описания констант. Список значений элементов при этом заключается в круглые скобки.

Например:

```
Const a: array [-1..1] of byte = (0,0,0);          { одномерный }
b: array [1..3, 1..2] of byte = ((1,2),(3,4),(5,6));{двумерный}
Months: array [1..12] of string [8] = ('январь', 'февраль',
                  'март', 'апрель', 'май', 'июнь', 'июль',
                  'август', 'сентябрь', 'октябрь',
                  'ноябрь', 'декабрь');
```

В этом случае компилятор не только выделяет память под массив, но и заполняет ячейки заданными значениями (в двумерном массиве – по строкам).

3.1. Базовые алгоритмы по работе с массивами

Перечислим некоторые действия над элементами массива:

1. ***Инициализация*** (присваивание начальных значений) массива.

```
For i := 1 to 4 do a[i] := 0;
```

Для двумерного массива – вложенный цикл:

```
For i := 1 to 10 do
      For j := 1 to 15 do b[i, j] := 0;
```

2. ***Ввод-вывод*** значений производится поэлементно:

```
For i := 1 to 10 do read (a[i]);
...
For i := 1 to 10 do write (a[i]:7);
```

Формирование массива путем ввода с клавиатуры – достаточно утомительное занятие. Для этого можно использовать массивы, описанные как типизированные константы (см. выше). При математическом моделировании и в играх для создания «случайной ситуации» массив заполняется с помощью генератора случайных

чисел:

```
Var a: array[1..10] of integer;
...
Randomize;
For i := 1 to 10 do a[i] := random(21);
```

3.***Копирование*** массивов – присваивание значений всех элементов одного массива всем элементам другого.

```
For i := 1 to 10 do b[i] := a[i];
```

4.***Поиск*** в массиве элементов, удовлетворяющих какому-либо условию.

Например, определить, сколько элементов массива равны нулю.

```
k := 0;
For i := 1 to 10 do
        if a[i] = 0 then k := k + 1;
```

5.***Перестановка*** значений элементов массива осуществляется с помощью дополнительной переменной.

Например, поменять местами первый и пятый элементы.

```
v := a[5];
a[5] := a[1];
a[1] := v;
```

6.***Сортировка*** массива.

Для решения многих задач необходимо упорядочить данные по определенному признаку (например, в порядке возрастания или убывания значений). Такая работа называется сортировкой данных.

Рассмотрим простейшую сортировку – обменную (метод «пузырька»). На каждом шаге алгоритма рассматриваем два рядом стоящих элемента, начиная с первого (или последнего). Если очередная пара нарушает требуемый порядок, ее элементы меняют местами. Шаги повторяют до тех пор, пока очередной проход не вызовет ни одного обмена.

```
For k := 1 to n - 1 do {Цикл по номеру просмотра}
  For i := 1 to n - k do
   If a[i] > a[i+1] then
     begin
       w := a[i];
       a[i] := a[i+1]; {Перестановка элементов}
       a[i+1] := w;
     end;
```

Для больших массивов, содержащих тысячи элементов, применяют «быстрые» способы сортировки.

Попытаемся понять, как работать с массивом, на примере следующей задачи:

> *Имеется 10 чисел. Необходимо найти их сумму.*

```
const n = 10;
Var s, i: integer;
   a: array [1..n] of integer;              {описание массива}
Begin
   For  i := 1  to  n  do  read (a[i]);     {Ввод элементов массива}
   s := 0;
   for  i := 1  to  n  do  s := s + a[i];   {Нахождение суммы}
   writeln (s);
End.
```

В данной задаче все элементы массива вводились с клавиатуры. Но ввод элементов массива, как мы уже говорили, можно осуществлять и по-другому: задавая весь массив в разделе констант.

Решим задачу вторым способом.

```
const n = 10;                          {Массив задан как константа}
     a: array [1..n] of integer = (1, 0, -2, 7, 8, 9, 9, 4, 0, -4);
Var s, i: integer;
Begin
   writeln ('Введённый массив');
   for  i := 1  to  n  do  write (a[i], ' ');  {Вывод массива}
   s := 0;
   for  i := 1  to  n  do  s := s + a[i];      {Нахождение суммы}
   writeln;
   writeln (s);
End.
```

Вопросы для повторения:

1. Что такое массив?
2. Что такое элемент массива?
3. Что такое индекс массива?
4. Что называется базовым типом?
5. Зачем нужны массивы?
6. Как можно обратиться к ячейке массива?
7. Какого типа могут быль элементы массива?
8. Какого типа может быть индекс массива?
9. Как можно осуществить ввод элементов массива? Назовите не менее трех способов.

Задания для самостоятельной работы:

1. Имеется целочисленный массив, состоящий из 15 элементов:
   a) найти сумму *a*[1] и *a*[7] элементов;
   b) найти разность *a*[9] и *a*[3] элементов;
   c) найти среднее арифметическое всех элементов массива;
   d) найти произведение всех элементов массива.
2. Имеется целочисленный массив, состоящий из *N* элементов (*N* - чётное):
   a) найти сумму первых *N*/2 элементов;
   b) найти сумму элементов стоящих на чётных позициях;
   c) найти произведение элементов стоящих на нечётных позициях;
   d) найти сумму чётных элементов стоящих на чётных позициях.
3. Имеется массив, состоящий из 9 элементов. Найти сумму, среднее арифметическое и произведение всех элементов массива.

3.2. Поиск в массиве элементов с заданными свойствами

Рассмотрим несколько примеров:

> *Имеется **n** целых чисел. Необходимо найти число, равное **K** (элемент последовательности, значение которого равно **K**). Если такой элемент в последовательности есть, то указать его порядковый номер.*

```
Var k, i, n, p: integer;
    a: array [1..30] of integer
Begin
   readln (n);
   for  i := 1  to  n  do readln (a[i]);
   readln (k);
   p := 0;
```

```
   for  i := 1  to  n  do
     if  a[i] = k  then  p := i;
   if   p = 0
      then  writeln ('элемента в таблице нет')
      else  writeln ('элемент найден, индекс =', p)
End.
```

В данной задаче в случае наличия в таблице нескольких элементов, равных *K* будет выдан наибольший индекс. Для поиска первого такого элемента в таблице используют цикл:

```
     i := 1;
     While (i <= n) and (a[i] <> k) do  i := i + 1;
```

> *Имеется **n** вещественных чисел. Необходимо наибольший элемент и указать его порядковый номер.*

```
Var a: array [1..30] of real;
    max: real;
    j, i: integer;
Begin
   readln(n);
     ..                                {Ввод элементов массива}
   max := a[1];
   j := 1;
   for  i := 2  to  n  do
     if   max < a[i]   then
         begin                         {Поиск максимума}
           max := a[i];
           j := i;
         end;
   writeln;
   writeln ('Макс. элемент массива = ', max:6:3);
   writeln ('Индекс макс. элемента = ', j);
End.
```

Алгоритм:

1. Условно считаем первый элемент наибольшим (максимальным). Запоминаем его значение (`a[1]`) и его индекс (1).
2. Сравниваем значение максимального с очередным элементом таблицы (`i`), начиная со второго элемента и до последнего.
3. При нахождении элемента с большим значением меняем значение максимального на найденный элемент и запоминаем его индекс.
4. При наличии в таблице нескольких минимальных и при использовании условия `max < a[i]` будет найден максимальный элемент с наименьшим индексом (первый встретившийся в таблице), а если условие будет записано как `max <= a[i]`, то – с большим индексом.

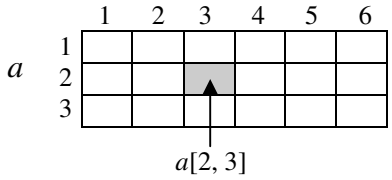
Задания для самостоятельной работы:

1. Имеется целочисленный массив, состоящий из 15 элементов. Найти минимальный элемент и его индекс.
2. В массиве хранится информация о количестве осадков (целые числа), выпавших за каждый день прошедшей недели. Вывести номера дней, когда осадков не было.
3. Дан массив целых чисел из *n* элементов. Найти и вывести номера элементов, заканчивающихся цифрой 0.

4. Рост *N* учеников класса представлен в виде массива. Найти количество учеников, рост которых не превышает значения *R*.
5. В массиве записаны результаты *N* игр футбольной команды (если игра закончилась выигрышем данной команды, то записано число 3, проигрышем - число 2, вничью - 1). Определить количество выигрышей, проигрышей, ничьих.
6. В массиве хранится информация о росте *N* человек. Определить, на сколько рост самого высокого человека превышает рост самого низкого.
7. В массиве хранится информация о стоимости 1 кг *N* видов конфет. Определить порядковый номер самого дешевого вида конфет. Если таких несколько, то должен быть найден индекс:
   а) первого из них;
   б) последнего из них.
8. В массиве хранится информация о стоимости каждой из *M* книг. Определить количество самых дешевых книг (с одинаковой минимальной ценой).

3.3. Двумерные массивы

Двумерные массивы имеют строки и столбцы. Элемент массива задается номером строки и номером столбца, на пересечении которых он находится.



a: **array** [1..3, 1..6] **of** <тип>;

Если количество строк равно количеству столбцов, массив называется квадратным, в противном случае – прямоугольным. Про массив, имеющий *m* строк и *n* столбцов, говорят, что он имеет размер *m**n*.

Вывод значений двумерного массива в виде таблицы:

```
For i := 1 to m do
      For j := 1 to n do read (b[i, j]);
...
For i := 1 to m do
      begin
          For j := 1 to n do write (b[i, j]:7);
          writeln();
      end;
```

🗎 **Рассмотрим несколько примеров:**

> *Задан массив размера n строк и m столбцов. Заполнить его целыми числами. Распечатать по строкам. Найти и вывести сумму всех элементов и их среднее арифметическое значение.*

```
Var i, j, s, m, n: integer;
    sr: real;
    a: array [1..10, 1..20] of integer;
                                                    {описание массива}
Begin
  readln (n, m);
  for  i := 1  to  n  do
   for  j := 1  to  m  do readln (a[i, j]);        {Ввод массива}
   s := 0;
   for  i := 1  to  n  do
     for  j := 1  to  m  do s := s + a[i, j];   {Нахождение суммы}
   writeln ('Сумма = ', s);
   sr := s / (n * m);                            {Нахождение среднего}
   writeln ('Среднее = ', sr)
End.
```

> *Среди столбцов с нечетными номерами в заданной целочисленной матрице размерностью n*m найти столбец с максимальной суммой модулей элементов.*

```
Var  i, j, m, n, k, s, max: integer;
     a: array [1..10, 1..20] of integer;
Begin
   Readln ( n,m);
{Ввод массива}
```

```
{Вывести таблицу по строкам}
   s := 0;
    for i := 1 to n do s := s + abs ( a[i, 1]); {Нахождение суммы
                                                 первого столбца}
                          {Первоначально будем считать эту сумму максимальной}
   max := s;
   k := 1;   j := 3;
   while   j <= m do
      begin
         s := 0;
         for  i := 1  to  n  do
             s := s + abs(a[i, j]);
         if  s > max
             then  begin
                     max := s;
                     k := j;
                   end;
         j := j + 2;
                                  {Переход к очередному нечетному индексу столбца}
      end;
   writeln ('Номер столбца с максим. суммой модулей элементов = ',
            k, 'знач. макс. = ', max);
end.
```

Для квадратной матрицы (количество строк равно количеству столбцов) введем некоторые понятия:

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

Главная диагональ – элементы a_{11}, a_{22}, a_{33}, a_{44} (на рисунке закрашены черным цветом). Индексы элементов, расположенных на главной диагонали, равны ($\boldsymbol{i = j}$).

Побочная диагональ – элементы a_{41}, a_{32}, a_{23}, a_{14} (на рисунке закрашены серым цветом). Сумма индексов элементов на единицу больше размерности массива ($\boldsymbol{i + j = n + 1}$).

	a_{12}	a_{13}	a_{14}
a_{21}		a_{23}	a_{24}
a_{31}	a_{32}		a_{34}
a_{41}	a_{42}	a_{43}	

Для элементов, расположенных над главной диагональю, $\boldsymbol{i < j}$.
Для элементов, расположенных под главной диагональю, $\boldsymbol{i > j}$.
Для элементов, расположенных над побочной диагональю, $\boldsymbol{i + j < n + 1}$.
Для элементов, расположенных под побочной диагональю, $\boldsymbol{i + j > n + 1}$.

Задания для самостоятельной работы:

1. Имеется целочисленный массив из $n*m$ элементов. Найти номер строки с минимальной суммой модулей элементов.
2. Имеется целочисленный массив из $n*m$ элементов. Найти индексы максимального элемента таблицы и значение максимума.
3. Имеется целочисленный массив из $n*m$ элементов. Каких элементов в массиве больше отрицательных или положительных?
4. Имеется целочисленный массив из $n*m$ элементов. Сообщить есть ли в таблице отрицательные элементы.
5. Имеется целочисленный массив из $n*m$ элементов. Найти среднее арифметическое элементов массива. Определить и вывести количество элементов массива, значение которых превышает это среднее значение.

Глава 4. Функции и процедуры

В программировании весьма типичной является ситуация, когда приходится выполнять одни и те же действия при разных исходных данных. Мы имеем некоторый алгоритм, который предназначен для решения подзадачи, выделенной из основной задачи (например, нахождение площади треугольника). Выписывать алгоритм для решения подзадачи каждый раз заново в том месте программы, где он понадобился, нерационально.

Pascal позволяет выделить любой алгоритм, описывающий подзадачу, и записать его только один раз, дав ему имя, а затем уже использовать его столько раз, сколько необходимо. В этом заключается смысл создания и использования ***подпрограмм***.

В Pascal подпрограммы подразделяются на ***процедуры*** (procedure) и ***функции*** (function), они представляют собой законченные программные объекты.

4.1. Функции

Структура функции повторяет структуру программы. Описание функции располагается до раздела операторов программы.

```
Function <имя функции> [(<параметры>)]: <тип>;
   [<описание констант>;]
   [<описание типов>;]
   [<описание переменных>;]
   Begin
         <тело функции>;
   End;
```

Опишем функцию возведения целого числа *a* в степень *n*:

```
Function stepen(a, n: integer):integer;
                                        {a, n – формальные параметры}
Var an, i: integer;
Begin
  an := 1;
  for i := 1 to n do an := an * a;
  stepen := an
End;
```

Первая строчка – заголовок функции. В скобках – аргументы, называемые ***формальными параметрами*** функции, с указанием их типа. Далее в заголовке указывается ***тип значения функции***, ее результата.

Результат вычисления функции (значение функции) ***присваивается имени*** функции:

```
stepen := an;
```

Обращение к функции из программы записывается в виде имени функции, за которым следует заключенный в скобки список параметров через запятые. Вызов функции не может быть самостоятельным оператором, потому что возвращаемое значение нужно куда-то записывать. Зато оно может стать равноправным участником арифметического выражения. Например:

```
c := stepen(2, 5);      {2, 5 – фактические параметры}
y := 3 * stepen(x, 3) -  5 * stepen(x, 2) + 0.61;
```

Параметры в обращении к функции называются ***фактическими*** параметрами. Фактическими параметрами могут быть не только константы и переменные, но также и выражения:

```
d := stepen(x + 2, 3);
```

Общее требование: формальные и фактические параметры должны совпадать по количеству, типу и порядку перечисления.

Функция может иметь несколько исходных данных (параметров), но ***только одно*** результирующее значение. Если надо в программе выделить самостоятельные части, которые на выходе дают несколько значений (результатов), то тогда используется ***процедура***.

4.2. Процедуры

Структура процедуры, как и структура функции, повторяет структуру программы. Описание процедуры располагается до раздела операторов программы.

```
Procedure <имя процедуры> [(<параметры>)];
      [<описание констант>;]
      [<описание типов>;]
      [<описание переменных>;]
      Begin
        <тело процедуры>;
      End;
```

Процедура ничего не возвращает явным образом, поэтому ее вызов является отдельным оператором в программе.

Замечание: После того как вызванная процедура завершит свою работу, управление передается оператору, следующему за оператором, вызвавшим эту подпрограмму.

Опишем процедуру возведения числа *b* в степень *k*:

```
Var b: real;
  k: integer;          {глобальные переменные}
  P: real;
Procedure power (a: real; n: integer);
                                        {a, n – формальные переменные}
Var i: byte;           {i – локальная переменная}
begin
  P := 1;
  For i := 1 to n do P := P * a;
end;

Begin
  ...
  power (3, 4);        {3, 4 – фактические параметры}
  ...
  readln (b, k);
  power (b, k);        {b, k – фактические параметры}
  writeln (b:4:2, ' в степени ', k, ' равно ', P)
End.
```

В теле основной программы объявлены переменные, для которых отводится память на весь период выполнения программы. Такие переменные называются ***глобальными***, поскольку их можно использовать в ***любой*** подпрограмме.

Для переменных, которые объявляются в разделе описания подпрограммы, память отводится только на время выполнения подпрограммы, поэтому они называются ***локальными***. Основная программа их «не замечает». Если попробовать обратиться к этим переменным в теле основной программы, то на этапе компиляции будет выдано сообщение о том, что данный идентификатор неизвестен.

Обратите внимание! Результат заносится в некоторую глобальную переменную (*P*).

Поскольку глобальные переменные видны в контекстах всех блоков, то их значение может быть изменено изнутри любой подпрограммы. Этот эффект называется

побочным, а его использование очень нежелательно, потому что может стать источником непонятных ошибок в программе.

Чтобы избежать побочного эффекта, необходимо строго следить за тем, чтобы подпрограммы изменяли только свои локальные переменные.

Для ликвидации этого неудобства используются параметры-переменные.

4.3. Использование параметров-переменных

Параметр-переменная обозначает в списке параметров переменную, в которую должен быть занесен ответ. Перед параметром-переменной стоит служебное слово **Var**.

Необходимо помнить, что при вызове процедуры параметры-переменные пусты, хотя для них отводится место в памяти. Для того чтобы они получили значение, его необходимо занести в параметры-переменные с помощью оператора присваивания.

Изменим процедуру power, вычисляющую a^n, так, чтобы результат вычисления поступал в основную программу через параметр-переменную.

```
Var a: real;
   n: byte;
   x, y: real;
Procedure power (b: real; k: byte; Var  c: real);
                                  {c – формальный параметр-переменная}
Var i: byte;
begin
   c := 1;
   For i := 1 to k do c := c * b;
end;

Begin
...
 power (7.2, 5, x);  {x – фактический параметр-переменная}
 writeln (x:3:4);
     ...
 power (-2.3, 4, y); {y – фактический параметр-переменная}
 writeln (y:4:2)
 End.
```

🗎 **Рассмотрим пример:**

Проанализируйте приведенную ниже программу. Определите, что будет выведено на экран в первом и во втором случае. Чем объяснить разницу? На данном примере сформулируйте различия между параметрами-значениями и параметрами-переменными.

```
Var k: integer;
Procedure plus1 (n: integer);
                                                  {параметр-значение}
   begin
      n := n + 10
   end;
Procedure plus2 (Var n: integer);
                                                  {параметр-переменная}
   begin
      n := n + 10
   end;
Begin
   k := 0;
   plus1(k);
```

```
    writeln(k);
    k := 0;
    plus2(k);
    writeln(k);
 End.
```

В первом случае будет напечатан 0.

Так как в описании процедуры используется параметр-значение, то значение фактического параметра передается «в одну сторону» – из основной программы в подпрограмму. Любые изменения такого параметра в подпрограмме никак не сказываются на значении соответствующего фактического параметра в основной программе.

Во втором случае будет напечатано 10.

Так как в описании процедуры используется параметр-переменная, то значение фактического параметра передается «в обе стороны» – из основной программы в подпрограмму и обратно. Любые изменения такого параметра в подпрограмме сразу же сказываются на значении фактического параметра в основной программе.

Заметим, что если фактическими параметрами для параметров-значений могут быть константы, имена переменных, выражения, то фактическими параметрами для параметров-переменных могут быть ***только имена переменных***.

Вопросы для повторения:

1. Дайте определение подпрограммы.
2. Как называются параметры, определяемые в заголовке подпрограммы?
3. Какие параметры называются фактическими?
4. Могут ли фактические параметры быть выражениями?
5. Чем синтаксически отличается описание процедуры от описания функции?
6. Какое количество значений возвращает функция?
7. Как определить тип значения, возвращаемого функцией?
8. Какие переменные называются локальными?
9. Чем глобальные переменные отличаются от локальных?
10. Наличие какого оператора необходимо для возвращения значения из функции в вызывающую программу?

Задания для самостоятельной работы:

1. Пусть дано n треугольников, заданных координатами своих вершин. Найдите треугольник с максимальной площадью. Напишите функцию для нахождения площади одного треугольника.
2. Пусть дано n произвольных четырехугольников, заданных координатами своих вершин. Найдите четырехугольник с максимальной площадью. Используйте функцию для вычисления площади четырехугольника.
3. Пусть дано n треугольников. Посчитайте количество треугольников, лежащих в каждой координатной четверти и не пересекающих оси координат. Используйте функцию для определения координатной четверти, в которой лежит треугольник.
4. Для двух представленных ниже квадратных уравнений определите, имеют ли они одинаковые корни. Напечатайте те корни уравнений, которые не совпадают.

$$a_1x^2 + b_1x + c_1 = 0,$$

$$a_2x^2 + b_2x + c_2 = 0.$$

5. * Напишите процедуру сложения двух дробей, результатом которой является несократимая правильная дробь. Используйте функцию для нахождения наибольшего общего делителя.

Глава 5. Символьный тип данных

До сих пор рассматривались исключительно программы, предназначенные для обработки числовых данных. Язык Pascal позволяет обрабатывать также символьные данные. Символьные данные могут быть либо константами, либо переменными. Следовательно, мы можем представлять в программах тексты и выполнять над ними некоторые операции. Значением символьного типа является множество всех символов ПК – цифры, буквы, знаки операций, специальные символы и т.д.

Каждому символу приписывается целое число в диапазоне 0..255. Это число служит кодом внутреннего представления символа.

Для кодировки используется код ASCII (American Standart Code for Information Interchange – американский стандартный код для обмена информацией).

Идентификатором символьного типа является зарезервированное слово `Char`, например:

```
Var c, ch: char;
```

Символьные значения можно вводить и выводить, присваивать, сравнивать. Из двух символов большим считается тот, код которого больше (в таблице ASCII он встречается позже).

В программе значения переменных и констант символьного типа должны быть заключены в *апострофы*. Например:

```
ch := '*';    a := '3';    Letter := 'G';
```

Кроме того, значения можно задать указанием кода:

```
k := #65;
```

Стандартные функции по работе с символьным типом данных

Функция	Действие	Тип аргумента	Тип результата	Пример
`Chr(x)`	Возвращает символ по его коду	`byte`	`char`	**`Chr(129) = 'Б'`**
`Ord(ch)`	Возвращает код по символу	`char`	`byte`	**`Ord('F') = 70`**
`Pred(ch)`	Возвращает предыдущий символ	`char`	`char`	**`Pred('Б') = 'А'`**
`Succ(ch)`	Возвращает следующий символ	`char`	`char`	**`Succ('F') = 'G'`**
`UpCase(ch)`	Если ch – строчная латинская буква, то возвращает прописную, в противном случае – сам символ	`char`	`char`	**`UpCase('s') = 'S'`** **`UpCase('ф') = 'Ф'`**

Рассмотрим следующие примеры:

Вывести на экран последовательность символов:
a
ab
abc
...
abc...xyz

Так как множество символов упорядоченно, то его можно использовать как параметр цикла.

```
Var c, d: char;
Begin
   For c := 'a' to 'z' do
      begin
         for d := 'a' to c do write (d);
         writeln
      end
End.
```

В вводимой с клавиатуры последовательности символов заменить все восклицательные знаки вопросительными. Признак конца ввода - /.

```
Var c: char;
Begin
   read (c);
   while c <> '/' do
      begin
         if c = '!' then write ('?')
               else write (c);
         read (c)
      end
End.
```

Вопросы для повторения:

1. Каково множество значений символьного типа?
2. Какой объем памяти требуется для хранения переменной символьного типа?
3. Что такое код символа?
4. Можно ли к данным символьного типа применять операции отношения?
5. Какие операции применимы к символьным данным?
6. Какие встроенные функции можно применить к символьным данным?
7. Верно ли, что `ord(chr(i)) = i`, где *i* – переменная целого типа?

Задания для самостоятельной работы:

1. Вывести на экран последовательности символов:
   а) abbccc...zz...z
   б) zyyxxx...aa...a
   в) abc...zbc...zc...z...z
2. В вводимой с клавиатуры последовательности символов посчитать количество букв «о». Признак конца ввода – точка.
3. Определить, есть ли во вводимой последовательности символов хотя бы одна цифра. Конец ввода – точка.
4. Составить программу, которая сообщает, какая буква `'A'` или `'B'` встречается чаще во введенной последовательности символов. Конец ввода – точка.
5. * В первой строке дано число n. Во второй строке дана последовательность символов, имеющая следующий вид: $d_1 \pm d_2 \pm ... \pm d_n$ (d_i – цифры, $n > 1$). Вычислить значение этой алгебраической суммы.

Глава 6. Строковый тип данных

При решении лексических и синтаксических задач мы используем данные символьного типа. Но данный тип не совсем удобен, так как позволяет хранить в переменной только один символ. Желательно иметь инструмент для задания целых последовательностей символов. Таким инструментом являются строки.

Идентификатором строкового типа является зарезервированное слово `String`, например:

```
Var st, str: string;
  str2: string[15];
```

Переменная, объявленная как переменная типа `String`, содержит от 0 до *N* символов, но не может превышать 255. Длину строки можно указать в квадратных скобках рядом со служебным словом `String`. В том случае, если длина строки не указана, она принимается максимально возможной, то есть 255 символов.

Тип `String` во многом похож на одномерный массив символов. К любому символу в строке можно обратиться точно так же, как к элементу массива, то есть указав рядом с именем переменной типа `String` в квадратных скобках индекс (порядковый номер) символа в строке.

Например:

```
Var st: string;
  ...
  if st[5] = 'A' then ...
```

Самый первый байт в строке имеет индекс, равный 0, и содержит текущую длину строки. Первый символ строки занимает второй байт и имеет индекс 1.

Работая с отдельными символами строки, надо соблюдать осторожность, так как `St[i]` лишь тогда означает *i*-ый символ строки, когда сама строка длиннее, чем *i*.

При использовании в выражениях строка заключается в *апострофы*.

Среди всевозможных значений строк есть пустая строка. Она изображается двумя апострофами, между которыми ничего нет – `''`.

Апостроф служит ограничителем строки. Чтобы не лишиться возможности иметь этот символ в составе строки, договорились повторять его там дважды. Например, оператор

```
write ('им''я')
```

выведет на экран `им'я`.

Над строковыми данными допустимы операции *сцепления* и *отношения*.

1. Операция сцепления строк (конкатенация). Обозначается эта операция знаком + (но это не сложение!).

   ```
   st := 'a' + 'b';
   st := st + 'c';      {st теперь содержит 'abc'}
   ```

   Длина результирующей строки не должна превышать 255 или заданной длины, в противном случае «лишние» символы отбрасываются.
2. Операции отношения (=, <>, >=, >, <=, <).

   Сравнение строк производится посимвольно, слева направо, до первого несовпадающего символа в соответствии с ASCII кодами символов.

   ```
   'MS-DOS' < 'MS-Dos'
   'program' > 'PROGRAM'
   ```

   Если строки различной длины, но в общей части совпадают, то более короткая строка меньше, чем более длинная.

   ```
   'Turbo' < 'Turbo Pascal'
   ```

Стандартные процедуры и функции по работе со строковым типом данных

Процедура или функция	Тип	Тип параметров	Тип значения	Действие
Concat(St1, ..., StN)	функция	все string	string	Строки объединяются в указанном порядке. Если результат превысит заданную длину или 255 символов, то «лишние» символы отбрасываются.
Copy(St, Poz, n)	функция	St: string; Poz, n: integer;	string	Из строки St, начиная с позиции Poz, выбирается n символов.
Delete(St, Poz, n)	процедура	St: string; Poz, n: integer;	–	Удаляются n символов из строки St, начиная с позиции Poz
Insert(SubSt, St, Poz)	процедура	SubSt: string; St: string; Poz: integer;	–	В строку St, начиная с позиции Poz, вставляется строка SubSt
Length(St)	функция	St: string;	integer	Определяется текущая длина строки St.
Pos(SubSt, St)	функция	SubSt: string; St: string;	integer	В строке St отыскивается первое вхождение строки SubSt (номер позиции). Если вхождения нет, результат равен 0.
Str(A, St)	процедура	A: integer; или A: real; St: string;	–	Число A преобразуется в строку, результат заносится в St
Val(St, A, Code)	процедура	St: string; A: integer; или A: real; Code: integer;	–	Если строка St состоит из цифр, то они преобразуются в числовое значение переменной A, значение Code равно 0. В противном случае преобразование не выполняется, значение Code равно номеру позиции в строке St, где обнаружен ошибочный символ, значение A не меняется.

Примеры:

1. Конкатенация строк

```
St1 := 'Turbo';
St2 := '-';
St3 := 'Pascal';
St := Concat(St1, St2, St3);
```

Значение строки `St` – `'Turbo-Pascal'`.

2. Копирование части строки

```
St := 'abcdef';
St1 := copy(St, 2, 3);
```

Значение строки `St1` – `'bcd'`.

3. Удаление части строки

```
St := 'abcdef';
Delete(St, 2, 3);
```

Значение строки `St` – `'aef'`.

4. Вставить подстроку в данную строку

```
St := 'ЭВМ 1841';
Subst := ' ЕС';
Insert(SubSt, St, 4);
```

Значение строки `St` – `'ЭВМ ЕС 1841'`.

5. Определение длины строки

```
St := 'Turbo Pascal';
k := Length(St);
```

Ответ: `k` = 12.

6. Поиск подстроки в строке

```
St := 'Иванов Александр г.Москва';
k := Pos('г.Москва', St);
```

Ответ: `k` = 18.

7. Преобразование числа в строку

```
a := 345;
Str(a, St);
```

Значение `St` – `'345'`.

```
b := 12.6789;
Str(b, St1);
```

Значение `St1` – `'1.2678900000E+01'`.

```
Str(b:2:2, St2);
```

Значение `St2` – `'12.67'`.

8. Преобразование строки в число

```
St := '     345';
Val(St, a, Code);
```

Ответ: `a` = `345`, `Code` = 0.

```
St := '12.6789';
Val(St, b, Code);
```

Ответ: `b` = `1,2678900000E+01`, `Code` = 0.

```
St := '345     ';
Val(St, c, Code);
```

Ответ: `c` – прежнее, `Code` = 4.

Рассмотрим следующие примеры:

Во введенной с клавиатуры строке символов подсчитать число букв ‘а’ и заменить их буквами ‘б’.

```
Var S: String;
   N: byte;
Begin
   Readln (S);
   N := 0;
   While Pos('a', S) > 0 do
      begin
         N := N + 1;
         S[Pos('a', S)] := 'б';
      end;
   Writeln ('В строке ', N, 'букв а');
   Writeln ('Получилась строка – ', S)
End.
```

Вопросы для повторения:

1. Какое максимально возможное количество символов может содержать строка?
2. Как при описании строкового типа указывается длина строки?
3. Всегда ли фактическая длина строки равна объявленной в описании?
4. Перечислите типовые операции над строками.
5. Может ли в процессе выполнения программы изменяться фактическая длина строки?
6. Может ли в процессе выполнения программы фактическая длина строки стать больше, чем объявлено в описании? Что произойдет в этом случае?
7. Как происходит сравнение строк:
   a. одинаковой длины?
   b. разной длины?
8. Перечислите основные процедуры и функции обработки строк.

Задания для самостоятельной работы:

1. Дана строка символов. Вывести ее на экран задом наперед.
2. Дана строка символов. Определить, является ли данная строка палиндромом.
3. Дана строка символов. Группы символов в ней между группами пробелов считаются словами. Подсчитать, сколько слов в строке.
4. Дана строка английских символов. Группы символов в ней между группами пробелов считаются словами. Подсчитать, сколько слов начинается на букву *b*.
5. Строка символов содержит только цифры. Вывести на экран номера позиций цифры 9.
6. * Дана строка символов. Выделить подстроку между первой и второй точкой.

Глава 7. Тестирование и отладка программ

Данная глава поможет начинающим программистам при создании и отладке своих первых программ, будет полезной она и для тех, кто считает себя уже знающим, опытным программистом.

Тестирование – один из наиболее трудоемких этапов создания программ. Если ваша программа выдала правдоподобный результат, то это еще не значит, что сама по себе она является правильной. Вполне возможно, что при повторном запуске с новым набором исходных данных программа выдаст неверный результат.

Одним из способов тестирования является тестирование по данным или тестирование с управлением по входу-выходу. В этом случае знания о внутренней структуре программы не учитываются, тестовые данные используются лишь в соответствии со спецификацией программы. При таком подходе для полного тестирования пришлось бы перебрать все возможные наборы входных данных, что, как правило, недостижимо из-за большого количества вариантов. Однако очень часто использование при прогоне даже ограниченного числа вариантов исходных данных «открывает глаза» на явно неверное в некоторых случаях поведение программы.

Другой способ тестирования – это тестирование, управляемое логикой программы. Оно позволяет исследовать внутреннюю структуру программы. При таком тестировании можно поступать по-разному. Представляется полезным ответить на вопросы, которые позволяют выявить специфические для языка Pascal ошибки. Разумеется, каждый программист пополняет для себя подобный список по мере накопления опыта.

Об *отладке* и *тестировании* создаваемых программ необходимо помнить, начиная с первых шагов разработки алгоритма решения задачи. Поэтому приведем здесь некоторые рекомендации по тестированию программ.

* Перед тестированием программы убедитесь в том, что в тексте программы всем переменным присваиваются значения. Экспериментальное тестирование программ может служить доказательством наличия ошибок, но никогда не докажет их отсутствия.
* Тесты нужно разрабатывать таким образом, чтобы проверить, корректно ли реагирует программа не только на допустимые входные данные, но и на заведомо неправильные.
* И в заключение – еще один практический совет: не исправляйте плохую программу ***– лучше перепишите ее.***

7.1. Чтобы программа была наглядной

1. При именовании величин *идентификаторы* следует выбирать так, чтобы они выражали смысловое значение этих величин. В дальнейшей работе это избавит автора программы или ее пользователя от необходимости всякий раз напрягать память для выяснения, что же этот идентификатор обозначает. Вместе с тем выбираемое имя не должно быть слишком длинным.
2. Программу следует разумно снабдить *комментариями*. Прежде всего, тексту программы должен предшествовать комментарий, дающий краткое, но полное описание ее функционального назначения, входных и промежуточных величин, характеристики алгоритма, фамилии автора и его координат. Комментарии бывают особенно полезными при описании и использовании подпрограмм: краткое описание их действий и характеристика формальных параметров сделают более понятными назначение подпрограммы.
3. Основой при разработке программы должен служить *структурный* подход, базирующийся на методе последовательного уточнения действий и использовании базовых управляющих структур: следования, ветвления и повторения. На начальных этапах проектирования допускается использование

абстрактных операторов, последующая детализация которых приводит к созданию подпрограмм. Не следует спешить писать в терминах операторов Pascal: уточнение действий должно быть постепенным. Это важно и потому, что на каком-то этапе может обнаружиться ошибка в выборе действия на предыдущем шаге, тогда придется переработать детализацию только этого действия. В противном случае пришлось бы «ворошить», а иногда просто заново разрабатывать всю программу.

4. *Расположение текста* программы на экране или на бумаге должно быть таким, чтобы подчеркивался структурный характер программы: следует выделять отступом от левого края строки те операторы или группы операторов, которые образуют вложенную часть другого составного оператора.
5. Приступая к очередному шагу уточнения действий, следует продумывать сразу *несколько вариантов* алгоритма. Это нужно, во-первых, чтобы выбрать наиболее оптимальный и подходящий, во-вторых, на случай, если придется внести изменения в проект программы на предыдущем шаге детализации.
6. Набирая программу на клавиатуре, следует иметь в виду, что буквы и символы, имеющие сходные изображения на экране на разных раскладках клавиатуры (например, буквы А в латинской и русской раскладках), по-разному представляются в памяти компьютера. В большинстве случаев замена одного символа другим, сходным с ним, приводит в дальнейшем к сообщениям компилятора об ошибках.
7. Рекомендуется также для большей наглядности и удобства отладки размещать на одной строке один оператор.

7.2. Ошибки в программах

Ошибки в программах разделяют на *синтаксические*, *семантические* и *логические*.

Ошибки первого типа выявляются *на стадии компиляции*, достаточно просто устраняются и говорят о несоответствии текста программы правилам языка Pascal.

Вторые возникают *на стадии выполнения* программы. Например, деление на ноль не устраняется на стадии компиляции, а приводит к ошибке на стадии выполнения, ибо зависит от конкретного значения переменной.

Третий тип ошибок – самый сложный в обнаружении. Программа работает так, как написана, но не так, как требуется. Такие случаи являются следствием многих причин. Главное, что эти ошибки сложно обнаружить. В системе Turbo Pascal имеется достаточно мощный отладчик, позволяющий находить ошибки третьего типа в программах. Он работает с исходным текстом программы, позволяет выполнять программу по шагам и отслеживать изменение переменных в процессе работы программы. Шагом при отладке является строка программы. Так, *если в строке исходного текста программы записано несколько операторов, то они будут выполнены за один шаг.* Однако если один несоставной оператор размещен на одной или нескольких строках, то он будет выполнен целиком.

Вопросы, которые нужно задать себе, чтобы избежать:

Ошибки обращения к данным

Существуют ли в программе обращения к переменным, которым не присвоены значения?

Если в программе используются массивы, то не выходят ли значения соответствующих индексов за границы, определенные при описании массива?

Ошибки вычисления

Не используются ли при некоторых вычислениях недопустимые типы данных?

Может ли возникнуть переполнение или потеря значимости во время вычисления выражения?

Может ли делитель в операторе оказаться равным нулю?

Может ли значение переменной выйти за пределы установленного для нее диапазона?

Правильно ли используются приоритеты операций для записи выражений?

Ошибки при сравнении

Нет ли в программе бесконечных циклов? Типичные ситуации:

- неверно сформулированное условие цикла (следует всегда помнить, что для цикла с предусловием `While` формулируется условие его выполнения, а для цикла с постусловием `Repeat` формулируется условие его прекращения);
- постоянное значение переменной цикла (`While`, `Repeat`).

Нет ли в программе циклов, которые никогда не будут выполняться?

Правильно ли расставлены операторные скобки `begin` и `end`? Ошибка в их расстановке может нарушить логику программы.

Ошибки интерфейса

Правильно ли заданы аргументы при вызове встроенных функций?

Не портит ли подпрограмма некоторые параметры и глобальные переменные, которые должны использоваться только как входные величины?

Глава 8. Практикум

8.1. Линейные алгоритмы

1. Даны два действительных числа. Найти среднее арифметическое этих чисел.
2. Дано трёхзначное число. Определить:
   а) сумму и произведение цифр числа;
   б) число, образованное перестановкой цифр исходного числа в обратном порядке;
   в) число, полученное перестановкой цифр сотен и десятков;
   г) число, полученное перестановкой цифр десятков и единиц.
3. Дано пятизначное число, записанное в двоичной системе счисления (СС). Переведите данное число в десятичную СС.
4. Дано четырёхзначное число. Получите двузначное число, удалив из исходного четырёхзначного числа цифры:
   а) тысяч и десятков (например: 2783 → 73);
   б) сотен и единиц (например: 2783 → 28);
   в) десятков и единиц (например: 2783 → 27);
   г) получите другие двузначные числа удалением цифр из исходного четырёхзначного числа.
5. Дано действительное число *a*. Не пользуясь никакими другими операциями кроме умножения, получить:
   а) a^6 за три операции;
   б) a^7 за четыре операции;
   в) a^9 за четыре операции;
   г) a^{28} за шесть операций;
   д) a^5 и a^{13} за пять операций;
   е) a^2, a^5 и a^{17} за шесть операций.
6. Дано действительное число *x*. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить $2x^4 - 3x^3 + 4x^2 - 5x + 6$. Разрешается использовать не более 4 умножений и 4 сложений и вычитаний.
7. Дано действительное число *x*. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить $1 - 2x + 3x^2 - 4x^3$ и $1 + 2x + 3x^2 + 4x^3$. Разрешается использовать не более восьми операций.
8. Дана сторона квадрата. Вычислите его периметр, длину диагонали и площадь.
9. Даны стороны *a* и *b* прямоугольника. Вычислите периметр, длину диагонали и площадь прямоугольника.
10. Дана сторона *a* равностороннего треугольника. Вычислите периметр и площадь треугольника.
11. Дана длина ребра куба. Вычислите диагональ куба, объём куба и площадь его боковой поверхности.
12. Три сопротивления *R1*, *R2*, *R3* соединены параллельно. Найдите сопротивление соединения.
13. Треугольник задан своими сторонами *a, b, c*. Используя формулу Герона ($s = \sqrt{p(p-a)(p-b)(p-c)}$, где p – полупериметр $p = (a+b+c)/2$), вычислите площадь треугольника.
14. Вычислите расстояние *d* между двумя точками с координатами *x1*, *y1*, *x2*, *y2* (расстояние $d = \sqrt{(x1-x2)^2 + (y1-y2)^2}$).
15. Треугольник задан координатами своих вершин *x1*, *y1*, *x2*, *y2*, *x3*, *y3*. Вычислите площадь треугольника.
16. Сумма первых n членов арифметической прогрессии вычисляется по формуле $S_n = (a_1 + a_n) \cdot n/2$, где $a_n = a_1 + d(n-1)$. Даны первый член прогрессии a_1 и разность

прогрессии d и количество членов прогрессии n. Вычислить S_n. Установить экспериментальным путём, при каком n значение S_n выходит за пределы integer. (*Арифметическая прогрессия - это последовательность чисел, в которой разность между двумя соседними элементами постоянна. Например: 3, 7, 11, 15 ,19, . . . Здесь 3 - первый член прогрессии (a_1), d=4 – разность прогрессии.*)

17. Задана температура в градусах по шкале Цельсия. Используя формулу перевода температуры из градусов по шкале Цельсия в градусы по шкале Фаренгейта $F = 1.8\ C + 32$, получите температуру по Фаренгейту.

8.2. Ветвление

1. Даны действительные числа x, y. Получить:
   b) max(x, y);
   c) min(x, y);
   d) max(x, y) и min(x, y).
2. Даны действительные числа x, y, z. Получить:
   a) max(x, y, z);
   b) min(x, y, z).
3. Задано натуральное число a. Является ли оно чётным?
4. Задано натуральное число a. Является ли оно кратным 9?
5. Дано трехзначное число. Кратна ли сумма его цифр семи?
6. Дано целое число $a > 9$. Меньше ли цифра десятков цифры единиц?
7. Дано трёхзначное число. Является ли сумма его цифр двузначным числом?
8. Дано три натуральных числа x, y, z. Вычислить количество чисел меньших 50.
9. Найти количество положительных чисел среди четырех заданных чисел a, b, c, d.
10. Даны два вещественных числа. Уменьшить второе число в пять раз, если оно больше первого по абсолютной величине.
11. Вводится название месяца. Вывести время года для этого месяца (январь – зима, …, март – весна, …, август – лето, …);
12. Составить программу, которая по введённому названию страны Европы будет выводить на экран название столицы (например, вводим Греция – получаем "Столица Греции – Афины");
13. Составить программу, которая по введённому на русском языке названию домашнего животного выведет перевод его на английский язык (например, вводим кот – получаем *cat*);
14. Придумайте программы-переводчики, энциклопедии, словари и др. по подобию предыдущего задания.
15. Заданы $x1$, $y1$, $x2$, $y2$ ($x1$, $y1$, $x2$, $y2 \neq 0$). Лежат ли точки ($x1$, $y1$) и ($x2$, $y2$):
    a) в одной четверти;
    b) в разных четвертях.
16. Заданы действительные x и y. Принадлежит ли точка (x, y) ветви параболы ($y = x^2$) лежащей во второй четверти?
17. Заданы два числа.
    a) является ли каждое из этих чисел чётным?
    b) является ли хотя бы одно из этих чисел чётным?
    c) является ли только одно из этих чисел чётным?
18. Заданы два натуральных числа. Является ли первое число двузначным, а второе однозначным?
19. Заданы два числа. Является ли первое число отрицательным, а второе положительным?
20. Дано двузначное число. Является ли сумма его цифр двузначным числом кратным трём.
21. Проверить, является ли число трехзначным, у которого цифры образуют геометрическую прогрессию (например: 139, 842).

22. Определить, является ли данное целое число N четным трёхзначным числом.
23. Даны действительные положительные числа x, y, z.
    a) Выяснить, существует ли треугольник с длинами сторон x,y,z.
    b) Если треугольник существует, то ответить – является ли он остроугольным.
24. Даны действительные положительные числа a, b, c, x, y. Выяснить, пройдёт ли кирпич с рёбрами a, b, c в прямоугольное отверстие со сторонами x и y. Просовывать кирпич в отверстие разрешается только так, чтобы каждое его ребро было параллельно или перпендикулярно каждой из сторон отверстия.
25. Дано натуральное число n ($n \le 9999$).
    a) Является ли это число палиндромом (перевёртышем) с учётом четырёх цифр, как, например, числа 2222, 6116, 0440 и т.д.?
    b) Верно ли, что это число содержит ровно три одинаковые цифры, как, например, числа 6676, 4544, 0006 и т.д.?
    c) Верно ли, что все четыре цифры числа различны?
26. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит восьми: первое число – номер вертикали, второе – номер горизонтали. Даны натуральные числа k, l, m, n, каждое из которых не превосходит восьми. Требуется:
    a) Выяснить, являются ли поля (k, l) и (m, n) полями одного цвета.
    b) На поле (k, l) расположен ферзь. Угрожает ли он полю (m, n)?
    c) Аналогично b), но ферзь заменяется конем.
    d) Выяснить, может ли ладья с поля (k, l) попасть за один ход на поле (m, n). Если нет, то выяснить, как это можно сделать за два хода (указать поле, на которое приводит первый ход).
    e) Аналогично d), но ладья заменяется ферзем.

8.3. Циклы

1. Даны действительное число a, натуральное число n. Вычислить:
    a) a^n;
    b) $a(a + 1) \ldots (a + n - 1)$.
2. Дано натуральное число n. Вычислить произведение первых n сомножителей:
    a) $\frac{1}{2} * \frac{3}{4} * \frac{5}{6} * \ldots$;
    b) $\frac{1}{1} * \frac{3}{2} * \frac{5}{3} * \ldots$.
3. Найти все двузначные числа, которые содержат цифру N.
4. Составьте программу возведения натурального числа в квадрат, используя следующую закономерность:
    $1^2 = 1$
    $2^2 = 1 + 3$
    $3^2 = 1 + 3 + 5$
    $4^2 = 1 + 3 + 5 + 7$
    . . .
    $n^2 = 1 + 3 + 5 + 7 + 9 + \ldots + (2n - 1)$.
5. Составить программу возведения заданного числа в третью степень, используя следующую закономерность:
    $1^3 = 1$
    $2^3 = 3 + 5$
    $3^3 = 7 + 9 + 11$
    $4^3 = 13 + 15 + 17 + 19$
    $5^3 = 21 + 23 + 25 + 27 + 29$
6. Среди двузначных чисел найти те, сумма квадратов цифр которых делится на заданное число n.

7. Написать программу поиска двузначных чисел, удовлетворяющих следующему условию: если к сумме цифр числа прибавить квадрат этой суммы, то получится само число.
8. Написать программу поиска трёхзначных чисел, квадрат которых оканчивается тремя цифрами, составляющими исходное число.
9. Написать программу поиска четырёхзначного числа, которое при делении на C даёт в остатке B, а при делении на B даёт в остатке D.
10. Найти сумму положительных нечётных чисел, меньших N.
11. Найти сумму целых положительных чисел из промежутка от A до B, кратных k (значения переменных A и B вводятся с клавиатуры).
12. Найти сумму целых положительных чисел, больших A, меньших B, кратных 3 и заканчивающихся на 2, 4 или 8.
13. В трёхзначном числе зачеркнули старшую цифру, когда полученное двузначное число умножили на 7, то получили данное число. Найти это число.
14. Сумма цифр трёхзначного числа кратна 7, само число также делится на 7. Найти все такие числа.
15. Среди четырёхзначных чисел выбрать те, у которых все четыре цифры различны.
16. Дано натуральное число. Найти все его делители и их сумму.
17. В 1626 году индейцы продали остров Манхеттен за 20$. Если бы эти деньги были помещены в банк на текущий счёт, и ежегодный прирост составил k%, то какова была бы сумма в текущем году?
20. Среди двузначных чисел найти те, которые делятся на число q, а сумма их цифр равна n ($0 < n \le 18$).
21. Найти минимальное число, большее N, которое нацело делится на K (K, N - натуральные числа).
22. Приписать по цифре 1 в начало и в конец записи числа n.
    (Например, ввод $n = 923$, вывод 19231).
23. Поменять местами первую и последнюю цифры числа.
    (Например, ввод $n = 9423$, вывод 3429).
24. Приписать к исходному числу n такое же число.
    (Например, ввод $n = 423$, вывод 423423).
25. Выяснить, сколько раз в натуральном числе встречается его максимальная цифра.
    (Например, ввод 4423, вывод 2 раза; ввод 9077, вывод 1 раз).
26. Выяснить, является ли разность максимальной и минимальной цифр числа чётной.
27. Дано натуральное число n. Требуется выяснить, можно ли представить его в виде суммы квадратов трёх натуральных чисел. Если можно, то:
    - указать тройку x, y, z таких натуральных чисел, что
      $x^2 + y^2 + z^2 = n;$
    - указать все тройки таких чисел, что $x^2 + y^2 + z^2 = n$.
28. Составить программу, печатающую k-ю цифру последовательности:
    - 12345678910 …, в которой выписаны подряд все натуральные числа;
    - 14916253649 …, в которой выписаны подряд квадраты всех натуральных чисел;
29. Составить программу для нахождения всех натуральных чисел n, m, k из интервала [a, b], удовлетворяющих соотношению $n^2 + m^2 = k^2$ (a и b заданы).
30. Стороны прямоугольника заданы натуральными числами M и N. Составить программу, которая будет находить, на сколько квадратов, стороны которых выражены натуральными числами, можно разрезать данный прямоугольник, если от него каждый раз отрезается квадрат максимально возможной площади.

8.4. Массивы

8.4.1. Одномерные массивы

1. Пусть дано N чисел. Определите, сколько среди них отличных от последнего.
2. Пусть дано N чисел. Напечатайте сначала все отрицательные из них, затем все остальные.
3. Пусть дано N вещественных чисел. Все отрицательные числа увеличьте на 0,5, а все неотрицательные, меньшие среднего арифметического, замените на 0,1.
4. Пусть даны целые числа a_1, …, a_{25}, b_1, …, b_{25}. Преобразуйте последовательность b_1, …, b_{25} по правилу, согласно которому если $a_i \le 0$, то b_i увеличивается в 10 раз, иначе b_i заменяется нулем (i = 1, …, 25).
5. Пусть даны вещественные числа a_1, …, a_{25}. Требуется умножить все члены этой последовательности на квадрат ее наименьшего члена, если $a_i \ge 0$, и на квадрат ее наибольшего члена, если $a_i < 0$.
6. Пусть дано N целых чисел. Получите новую последовательность, выбросив из исходной максимальный и минимальный элементы.
7. В целочисленной последовательности есть нулевые элементы. Создать массив из номеров этих элементов.
8. Пусть дано N целых чисел, каждое из которых отлично от нуля. Если в последовательности отрицательные и положительные члены чередуются, то ответом должна служить сама исходная последовательность. Иначе – получите все отрицательные члены последовательности, сохранив порядок их следования.
9. Пусть дано N различных целых чисел. Найдите среднее арифметическое чисел этой последовательности, расположенных между максимальным и минимальными числами (в сумму включить и оба этих числа).
10. Пусть даны координаты N точек на плоскости: x_1, y_1, …, x_n, y_n (n = 20). Найдите номера двух точек, расстояние между которыми наибольшее (считайте, что такая пара точек единственная).
11. Пусть в массиве содержатся результаты измерений температуры воздуха, которые проводились ежедневно в течение декабря месяца. Определите:
    a. среднемесячную температуру декабря;
    b. сколько раз температура была выше 0°С;
    c. день, когда первый раз температура поднялась выше нуля;
    d. сколько дней в декабре температура была выше средней;
    e. день, когда температура была ближе всего к средней температуре в декабре;
    f. минимальную температуру второй декады декабря;
    g. минимальную температуру тех дней декабря, которые следуют после последнего из самых теплых дней;
    h. среднюю температуру тех дней, которые предшествуют первому из самых холодных дней в декабре;
    i. любые два самых холодных дня;
    j. сколько раз температура в декабре меняла знак.
12. Пусть дано N натуральных чисел. Определите количество членов последовательности:
    a. являющихся нечетными числами;
    b. кратных 3 и не кратных 5;
    c. являющихся квадратами четных чисел;
    d. имеющих четные порядковые номера и являющихся нечетными числами.
13. Дана последовательность N действительных чисел. Выяснить, будет ли она возрастающей.
14. Пусть дано N целых чисел. Поменять местами наибольший и наименьший элементы.

15. При поступлении в вуз абитуриенты, получившие двойку на первом экзамене, ко второму не допускаются. В массиве записаны оценки *N* экзаменующихся, полученные на первом экзамене. Подсчитать, сколько человек не допущено ко второму экзамену.
16. Пусть дано *N* чисел, среди которых имеется один нуль. Вывести на печать все числа до нуля включительно.
17. У вас есть доллары. Вы хотите обменять их на рубли. Есть информация о стоимости купли-продажи в банках города. В городе *N* банков. Составьте программу, определяющую, какой банк выбрать, чтобы выгодно обменять доллары на рубли.
18. Пусть дано *N* целых чисел. Вычислить сумму элементов массива, порядковые номера которых совпадают со значением этого элемента.
19. Определить, сколько процентов от всего количества элементов последовательности целых чисел составляют нечетные элементы.

8.4.2. Двумерные массивы

1. Имеется целочисленный массив из $n*n$ элементов. Вычислить сумму и число положительных элементов, находящихся над главной диагональю.
2. Имеется массив из $n*m$ элементов. Найти в каждой строке максимальный и минимальный элементы и поменять местами с первым и последним элементом строки соответственно.
3. Упорядочить по возрастанию элементы каждой строки массива из $n*m$ элементов.
4. Имеется массив из $n*m$ элементов. Записать на место отрицательных элементов нули, а на место положительных – единицы.
5. Имеется массив размером $n*m$, все действительные элементы которого различны. В каждой строке выбрать элемент с наименьшим значением, затем среди этих чисел выбрать наибольшее. Указать индексы элемента с найденным значением.
6. Имеется массив размером $n*n$ (n – нечетное), все элементы которого различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.
7. Имеется массив из $n*n$ элементов и число k. Разделить элементы k-ой строки на элемент главной диагонали, расположенный в этой строке.
8. Дан массив из $n*m$ элементов. Найти наибольший и наименьший элементы массива и поменять их местами.
9. Дан массив из $n*m$ элементов. Найти строку с наибольшей и наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.
10. Дан массив из $n*m$ элементов. Найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

8.5. Функции и процедуры

1. Написать программу для нахождения наибольшего общего делителя четырех натуральных чисел.
2. Даны координаты вершин n-угольника. Определить его периметр. Вычисление расстояния между вершинами оформить подпрограммой.
3. Вычислить площадь правильного шестиугольника со стороной a, используя подпрограмму вычисления площади треугольника.
4. На плоскости заданы своими координатами n точек. Составить программу, определяющую, между какими из пар точек самое большое расстояние. *Указание*. Координаты точек занести в массив.
5. Написать программу, проверяющую, являются ли данные три числа взаимно простыми.
6. Составить программу для вычисления суммы факториалов всех нечетных чисел от 1 до 9.

7. Заменить отрицательные элементы одномерного массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество произведенных замен.
8. Составить программу, определяющую, в каком из данных двух чисел больше цифр.
9. Дано простое число. Составить функцию, которая будет находить следующее за ним простое число.
10. Составить программу вывода на экран всех натуральных чисел, не превосходящих N и делящихся на каждую из своих цифр.

8.6. Символьный тип данных

1. Подсчитайте, сколько раз среди последовательности символов встречается символ, задаваемый вводом. Количество вводимых символов также определяется вводом.
2. Пусть задан текст, за которым следует точка. Поменяйте все строчные латинские буквы, которые встречаются в тексте, на прописные.
3. Пусть вводится последовательность символов длиной не более 20. Признак конца последовательности – точка. Верно ли, что в последовательности содержится четное количество строчных латинских букв?
4. Пусть задан текст, состоящий из слов. Под словом понимается последовательность символов, не содержащая пробелов и знаков препинания. Если слово начинается с латинской строчной буквы, замените ее на прописную.
5. Дана строка символов. Группы символов в ней между группами пробелов считаются словами. Определить, сколько слов начинается и заканчивается одной и той же буквой.
6. Пусть дан непустой текст из заглавных латинских букв, за которым следует точка. Определите, упорядочены ли эти буквы по алфавиту.

8.7. Строковый тип данных

1. Дана строка, содержащая текст. Найти длину самого короткого и самого длинного слова.
2. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов ему предшествует.
3. Дана строка, содержащая текст, заканчивающийся точкой. Вывести на экран слова, содержащие три буквы.
4. Дана строка. Определить, сколько раз входит в нее группа символов *abc*.
5. Дана строка. Подсчитать, сколько различных символов встречается в ней. Вывести их на экран.
6. Дана строка. Подсчитать самую длинную последовательность подряд идущих букв *a*.
7. Имеется строка, содержащая буквы латинского алфавита и цифры. Вывести на экран длину наибольшей последовательности цифр, идущих подряд.
8. В строке между словами вставить вместо пробела запятую и пробел.
9. Удалить часть символьной строки, заключенной в скобки (вместе со скобками).
10. Строка содежит одно слово. Проверить, будет ли оно читаться одинаково справа налево и слева направо (т.е. является ли она палиндромом).
11. В записке слова зашифрованы – каждое из них записано наоборот. Расшифровать сообщение.
12. Результаты вступительных экзаменов представлены в виде списка из N строк, в каждой строке которого записаны фамилия абитуриента и отметки по каждому из M экзаменов. Определить количество абитуриентов, сдавших вступительные экзамены только на «отлично».
13. Из заданной символьной строки выбрать только те символы, которые встречаются в ней только один раз, в том порядке, в котором они встречаются в тексте.
14. В заданной строке удалить все лишние пробелы.

Список литературы

1. *Абрамов С. А., Зима Е. В.* Начала программирования на языке паскаль. – М.: Наука, 1987.
2. *Бондарев В. М., Рублинецкий В. И., Качко Е. Г.* Основы программирования. – Харьков: Фолио; Ростов н/Д: Феникс, 1997.
3. *Попов В. Б.* Turbo Pascal для школьников. Версия 7.0: Учеб. пособие. – 2-е изд., стереотип. – М.: Финансы и статистика, 1998.
4. *Молчанова С. И.* Основы программирования. Турбо-Паскаль 7.0 для школьников и абитуриентов. – М.: Издательство АСТ, 2000.
5. *Окулов С. М.* Основы программирования. – М.: ЮНИМЕДИАСТАЙЛ, 2002.
6. Турбо-Паскаль в примерах: Кн. Для учащихся 10–11 кл. / А. Б. Николаев, Л. А. Акатнова, С. В. Алексахин и др. – М.: Просвещение, 2002.
7. *Костюкова Н. И.* Знакомьтесь – Паскаль! Методические рекомендации и задачи по программированию. – Новосибирск: Сиб. Унив. Изд-во, 2003.
8. *Семакин И. Г.* Основы программирования: Учебник для сред. проф. образования / И. Г. Семакин, А. П. Шестаков. – 2-е изд., стер. – М.: Издательский центр «Академия», 2003.
9. Программирование на языке Паскаль: задачник / под ред. Усковой О. Ф. – СПБ.: Питер, 2003.
10. *Рапаков Г. Г., Ржеуцкая С. Ю.* Turbo Pascal для студентов и школьников. – СПб.: БХВ-Петербург, 2005.