Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего и профессионального образования

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет математики и информатики

И.А. Шпехт, Р.Р. Саакян

Информатика (алгоритмизация и основы программирования)

Учебно-методическое пособие

Благовещенск
2005

Печатается по решению
редакционно-издательского совета
факультета математики и нформатики
Амурского государственного
университета

И.А. Шпехт,
Р.Р. Саакян

Информатика (алгоритмизация и основы программирования):
Учебно-методическое пособие. / Благовещенск, Амурский гос. ун-т. 2005.

Пособие содержит краткие теоретические сведения об основах программирования на алгоритмическом языке Pascal, примеры решения задач и набор заданий для самостоятельного решения. Предназначено для студентов специальностей 230201 – «Информационные системы и технологии», 010501 – «Прикладная математика», 220301 – «Автоматизация технологических процессов и производств», изучающих начала программирования.

Рецензенты: З.Ф.Юсупов, заместитель директора по научной работе Благовещенского филиала Московской академии предпринимательства при Правительстве г.Москвы, доцент кафедры информатики и математики, канд.техн.наук, доцент;
Т.А.Галаган, доцент кафедры нформационных и управляющих систем Амурского государственного университета, канд.техн.наук.

В В Е Д Е Н И Е

Учебно-методическое руководство к лабораторным работам по курсу «Информатика» предназначено для студентов специальностей 230201 – «Информационные системы и технологии», 010501 – «Прикладная математика», 220301 – «Автоматизация технологических процессов и производств» ***Цель методического указания*** – закрепить теоретические знания студентов об основах составления алгоритмов решения задач, дать им практические навыки программирования на алгоритмическом языке Pascal.

Учебно-методическое указание включает 9 лабораторных работ. Каждая лабораторная работа содержит краткие теоретические сведения по определенной теме и варианты заданий.

Выполнение лабораторной работы складывается из нескольких стадий: подготовка, разработка программы, получение результатов, оформление и сдача отчетов по работе.

Оформление и сдача отчета по лабораторной работе. Каждую лабораторную работу студент оформляет в рабочей тетради в виде отчета, который должен содержать:

- исходные данные по вариантам;
- алгоритм решения задачи в виде блок-схемы;
- текст программы;
- результаты решения тестового примера.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 1
С Т А Н Д А Р Т Н Ы Е Т И П Ы Д А Н Н Ы Х

К стандартным относятся целые, действительные, логические, символьный и адресный типы.

К ЦЕЛЫМ типам относят:

Integer (диапазон значений: -32768 .. 32767),

Shortint (диапазон значений: -128 .. 127),

Byte (диапазон значений: 0 .. 255),

Word (диапазон значений: 0 .. 65535),

Longint (диапазон значений: -2147483648 .. 2147483647).

Над целыми операндами можно выполнять следующие арифметические операции: сложение, вычитание, умножение, деление, получение остатка от деления. Знаки этих операций:

+ - * div mod

Результат арифметической операции над целыми операндами есть величина целого типа. Результат выполнения операции деления целых величин есть целая часть частного. Результат выполнения операции получения остатка от деления - остаток от деления целых. Например:

17 div 2 = 8, 3 div 5 = 0.

17 mod 2 = 1, 3 mod 5 = 3.

Операции отношения, примененные к целым операндам, дают результат логического типа TRUE или FALSE (истина или ложь).

В языке ПАСКАЛЬ имеются следующие операции отношения:

Равенство **=,**

Неравенство **<>,**

Больше или равно **>=,**

Меньше или равно **<=,**

Больше **>,**

Меньше **<.**

К аргументам целого типа применимы следующие **стандартные (встроенные) функции,** результат выполнения которых имеет целый тип:

Abs(X), Sqr(X), Succ(X), Pred(X),

и которые определяют соответственно абсолютное значение **X, X в квадрате, X+1, X-1.**

Следующая группа **стандартных функций** для аргумента целого типа дает действительный результат:

Sin(X), Cos(X), ArcTan(X), Ln(X), Exp(X), Sqrt(X).

Эти функции вычисляют синус, косинус и арктангенс угла, заданного в радианах, логарифм натуральный, экспоненту и корень квадратный соответственно.

Результат выполнения **функции проверки целой величины на нечетность Odd(X)** имеет значение истина, если аргумент нечетный, и значение ложь, если аргумент четный:

X=5 Odd(X)=TRUE , X=4 Odd(X)=FALSE.

Для быстрой работы с целыми числами определены **процедуры:**

Inc(X) X:=X+1

Inc(X,N) X:=X+N

Dec(X) **X:=X-1**

Dec(X,N) **X:=X-N**

К ДЕЙСТВИТЕЛЬНЫМ типам относят:

Real (диапазон значений: 2.9e-39 .. 1.7e+38),

Single (диапазон значений: 1.5e-45 .. 3.4e+38),

Double (диапазон значений: 5.0e-324 .. 1.7e+308),

Extended (диапазон значений: 3.4e-4932 .. 1.1e+4932).

Над действительными операндами можно выполнять следующие **арифметические операции,** дающие действительный результат:

Сложение **+** ,

Вычитание **-** ,

Умножение ***** ,

Деление **/** .

К величинам действительного типа применимы все **операции отношения,** дающие булевский результат.

Один из операндов, участвующих в этих операциях, может быть целым.

К действительным аргументам применимы **функции,** дающие действительный результат:

Abs(X), Sqr(X), Sin(X), Cos(X), ArcTan(X), Ln(X), Exp(X), Sqrt(X), Frac(X), Int(X), Pi.

Функция **Frac(X)** возвращает дробную часть X, функция **Int(X)** – целую часть X.

Безаргументная функция **Pi** возвращает значение числа Пи действительного типа.

К аргументам действительного типа применимы также **функции**

Trunc(X) и Round(X),

дающие целый результат. Первая из них выделяет целую часть действительного аргумента путем отсечения дробной части, вторая округляет аргумент до ближайшего целого.

ЛОГИЧЕСКИЙ тип (Boolean) определяет те данные, которые могут принимать логические значения **TRUE** и **FALSE**.

К булевским операндам применимы следующие **логические операции:**

not and or xor.

Логический тип определен таким образом, что **FALSE < TRUE.** Это позволяет применять к булевским операндам все операции отношения.

СИМВОЛЬНЫЙ тип (Char) определяет упорядоченную совокупность символов, допустимых в данной ЭВМ. Значение символьной переменной или константы - это один символ из допустимого набора.

Символьная константа может записываться в тексте программы двумя способами:

- как один символ, заключенный в апострофы, например:

'A' 'a' 'Ю' 'ю';

-с помощью конструкции вида **#K,** где **K** - код соответствующего символа, при этом значение **K** должно находиться в пределах **0..255.**

К величинам символьного типа применимы все **операции отношения.**

Для величин символьного типа определены две **функции преобразования**

Ord(C) Chr(K).

Первая функция определяет порядковый номер символа **С** в наборе символов, вторая определяет по порядковому номеру **К** символ, стоящий на **К**-ом месте в наборе символов. Порядковый номер имеет целый тип.

К аргументам символьного типа применяются **функции,** которые определяют предыдущий и последующий символы:

Pred(C) Succ(C). Pred('F') = 'E' ; Succ('Y') = 'Z' .

При отсутствии предыдущего или последующего символов значение соответствующих функций не определено.

Для литер из интервала **'a'..'z'** применима функция **UpCase(C),** которая переводит эти литеры в верхний регистр **'A'..'Z'.**

О П Е Р А Т О Р Ы В В О Д А И В Ы В О Д А

Для ввода исходных данных используются операторы процедур ввода:

```
Read(A1,A2,...AK);
ReadLn(A1,A2,...AK);
ReadLn;
```

Первый из них реализует чтение **К** значений исходных данных и присваивание этих значений переменным **А1, А2, ..., АК.** Второй оператор реализует чтение **К** значений исходных данных, пропуск остальных значений до начала следующей строки, присваивание считанных значений переменным **А1, А2, ..., АК.** Третий оператор реализует пропуск строки исходных данных.

Переменные, образующие список ввода, могут принадлежать либо к целому, либо к действительному, либо к символьному типам. Чтение исходных данных логического типа в языке ПАСКАЛЬ недопустимо.

Операторы ввода при чтении значений переменных целого и действительного типа пропускает пробелы, предшествующие числу. В то же время эти операторы не пропускают пробелов, предшествующих значениям символьных переменных, так как пробелы являются равноправными символами строк. Пример записи операторов ввода:

```
var V, S: Real;
   W, J: Integer;
   C, D: Char;
.............
Read(V, S, W, J);
Read(C, D);
```

Значения исходных данных могут отделяться друг от друга пробелами и нажатием клавиш табуляции и Enter.

Для вывода результатов работы программы на экран используются операторы:

```
Write(A1,A2,...AK);
WriteLn(A1,A2,...AK);
WriteLn;
```

Первый из этих операторов реализует вывод значений переменных **A1, A2,...,AK** в строку экрана. Второй оператор реализует вывод значений переменных **A1, A2, ..., AK** и переход к началу следующей строки. Третий оператор реализует пропуск строки и переход к началу следующей строки.

Переменные, составляющие список вывода, могут относиться к целому, действительному, символьному или булевскому типам. В качестве элемента списка вывода кроме имен переменных могут использоваться выражения и строки.

Вывод каждого значения в строку экрана происходит в соответствии с шириной поля вывода, определяемой конкретной реализацией языка.

Форма представления значений в поле вывода соответствует типу переменных и выражений: величины целого типа выводятся как целые десятичные числа, действительного типа - как действительные десятичные числа с десятичным порядком, символьного типа и строки - в виде символов, логического типа - в виде логических констант **TRUE** и **FALSE**.

Оператор вывода позволяет задать ширину поля вывода для каждого элемента списка вывода. В этом случае элемент списка вывода имеет вид **А:К**, где **А** - выражение или строка, **К** - выражение либо константа целого типа.

Если выводимое значение занимает в поле вывода меньше позиций, чем **К,** то перед этим значением располагаются пробелы. Если выводимое значение не помещается в ширину поля **К,** то для этого значения будет отведено необходимое количество позиций. Для величин действительного типа элемент списка вывода может иметь вид **А:К:М,** где **А -** переменная или выражение действительного типа, **К** - ширина поля вывода, **М** - число цифр дробной части выводимого значения. **К** и **М** - выражения или константы целого типа. В этом случае действительные значения выводятся в форме десятичного числа с фиксированной точкой.

Пример записи операторов вывода:

```
. . . . . .
var A, B: Real;       P,Q:Integer;
   R, S: Boolean;    T, V, U, W: Char;
. . . . . .
WriteLn(A, B:10:2);
```

```
WriteLn(P, Q:8);
WriteLn(R:8, S:8);
WriteLn(T, V, U, W);
```

П Р О Г Р А М М Ы Л И Н Е Й Н О Й С Т Р У К Т У Р Ы

Программы линейной структуры реализуют линейные алгоритмы, при исполнении которых выполняются все действия подряд от начала до конца алгоритма.

Формально для реализации программы линейной структуры используются четыре оператора:

- оператор присваивания;
- оператор обращения к процедуре;
- составной оператор;
- пустой оператор.

На практике программы линейной структуры встречаются не так часто.

З А Д А Н И Я К Л А Б О Р А Т О Р Н О Й Р А Б О Т Е

1. Написать программу вычисления объема параллелепипеда. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление объема параллелепипеда.
Введите исходные данные:
Длина    (см)  ->  9
Ширина   (см)  ->  7.5
Высота     (см)  ->  5

Объем: 337.50 куб.см.
```

2. Написать программу вычисления площади поверхности параллелепипеда. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление площади поверхности параллелепипеда.
Введите исходные данные:
Длина   (см)  ->  9
Ширина  (см)  ->  7.5
Высота    (см)  ->  5

Площадь поверхности: 90.00 кв.см.
```

3. Написать программу вычисления объема цилиндра. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление объема цилиндра.
Введите исходные данные:
Радиус основания   (см)  ->  5
Высота цилиндра    (см)  ->  10

Объем цилиндра: 1570.80 куб.см.
Для завершения работы программы нажмите <Enter>.
```

4. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости покупки.
Введите исходные данные:
```

```
Цена тетради (руб.)          -> 2.75
Количество тетрадей       -> 5
Цена карандаша               -> 0.85
Количество карандашей  -> 2

Стоимость покупки: 15.45 руб.
```

5. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и такого же количества обложек к ним. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости покупки.
Введите исходные данные:
Цена тетради  (руб.)         -> 2.75
Цена обложки (руб.)         -> 0.5
Количество комплектов (шт.)  -> 7

Стоимость покупки: 22.75 руб.
```

6. Написать программу вычисления стоимости поездки на автомобиле на дачу (туда и обратно). Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости поездки на дачу и обратно.
Расстояние до дачи (км) -> 67
Расход бензина (литров на 100 км пробега) -> 8.5
Цена литра бензина (руб.) -> 14.00

Поездка на дачу и обратно обойдется в 159.46 руб.
```

7. Написать программу пересчета расстояния из верст в километры (1 верста равняется 1066,8 км). Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Пересчет расстояния из верст в километры.
Введите расстояние в верстах и нажмите <Enter>.
-> 100

100 верст(а/ы) - это 106.68 км.
```

8. Написать программу пересчета величины временного интервала, заданного в минутах, в величину, выраженную в часах и минутах. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Введите величину временного интервала (в минутах)
и нажмите <Enter>.
-> 150

150 минут  -  это 2 ч. 30 мин.
```

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 2
УСЛОВНЫЙ ОПЕРАТОР
ПОЛНАЯ И НЕПОЛНАЯ РАЗВИЛКА
ОПЕРАТОР ВЫБОРА

На практике решение большинства задач не удается описать с помощью программ линейной структуры. Обычно после проверки некоторого условия выполняется та или иная последовательность операторов, однако при этом происходит нарушение естественного порядка выполнения операторов. Для этих целей используют управляющие операторы. Условный оператор, который используется для реализации разветвлений в программе, имеет следующую структуру

IF <логическое выражение> THEN <оп1>

ELSE <оп2>;

Если логическое выражение, выступающее в качестве условия, принимает значение **False,** то выполняются операторы, расположенные после **else (оп2),** если **True, —** операторы, следующие за **then**. Если количество операторов больше, чем 1, необходимо использовать операторные скобки **begin .. end.** При записи логического выражения следует избегать знака = **(равно)** для действительных переменных, так как они представляются неточно, а поэтому может не произойти совпадений значений выражений, стоящих слева и справа от знака равно. Для устранения указанного недостатка следует требовать выполнения условия с заданной точностью, т.е. вместо отношения $X = Y$ рекомендуется, например,

Abs(X - Y) < 1E-8.

Поскольку развилка может быть неполной, то возможна и неполная форма записи условного оператора:

IF <логическое выражение> THEN <оп>;

Условный оператор реализует разветвление вычислительного процесса по двум направлениям, одно из которых осуществляется при выполнении условия, другое — в противном случае. Для реализации разветвлений более

чем по двум направлениям необходимо использовать несколько условных операторов. Рассмотрим примеры.

Задача 1. Даны действительные числа ***x, y.*** Если ***x*** и ***y*** отрицательны, то каждое значение заменить модулем; если отрицательно только одно из них, то оба значения увеличить на **0,5;** если оба значения неотрицательны и ни одно из них не принадлежит отрезку **[0,5; 2,0],** то оба значения уменьшить в **10** раз; в остальных случаях ***x*** и ***y*** оставить без изменения.

Разработаем алгоритм решения задачи, после чего напишем программу.

Алгоритм запишем словесно:

```
1) ввести значения x, y;
2) если x<0 и y<0, найти их модули и перейти к п. 5,
иначе перейти к следующему пункту;
3) если x<0 или y<0, увеличить каждую величину на 0,5 и
перейти к п. 5,
   иначе перейти к следующему пункту;
4) если ни x, ни y не принадлежат отрезку [0,5; 2,0],
уменьшить их в 10 раз;
5) вывести значения x и y;
6) конец.
```

```
Program Usl_1;
 Var X, Y : Real;
 Begin
   Write('Введите два действительных числа ');
ReadLn(X, Y);
   If (X < 0) AND (Y < 0) THEN
                              begin
                                X := ABS(X);
                                Y := ABS(Y)
                              end
```

```
                              ELSE
 IF (X < 0) OR (Y < 0) THEN
                             begin
                               X := X + 0.5;
                               Y := Y + 0.5
                              end
                            ELSE
  IF NOT (((X >= 0.5) AND (X <= 2))
      OR ((Y >= 0.5) AND (Y <= 2)))
                           THEN
                             begin
                               X := X / 10;
                               Y := Y / 10
                             end;
 WriteLn('Результат:');WriteLn('X=',X:10:6);WriteLn('Y=
', Y:10:6)
End.
```

Задача 2. Дано действительное число ***a.*** Вычислить ***f(a)***, если

$$f(x)=\begin{cases} 0 \text{ при } x \le 0, \\ x^2 - x \text{ при } 0 < x \le 1, \\ x^2 - \sin \pi x^2 \text{ придругихx}. \end{cases}$$

```
Program Usl_2;
 Var A, F : Real;
 Begin
  WriteLn('Введите     действительное     число:     ');
ReadLn(A);
   IF A <= 0 THEN
               F := 0
             ELSE
              IF A <= 1 THEN
```

```
                    F := Sqr(A) - A
                  ELSE
                 F := Sqr(A) - SIN(Pi * Sqr(A));
 WriteLn('Значение функции F(x) при x = ',A:10:6,'
равно ',F:10:6);
End.
```

Кроме условного оператора в качестве управляющей структуры довольно часто используется оператор выбора **CASE**. Эта структура позволяет переходить на одну из ветвей в зависимости от значения заданного выражения (селектора выбора). Ее особенность состоит в том, что выбор решения здесь осуществляется не в зависимости от истинности или ложности условия, а является вычислимым. Оператор выбора позволяет заменить несколько операторов развилки (в силу этого его ещё называют оператором множественного ветвления).

В конструкции **CASE** вычисляется выражение ***K*** и выбирается ветвь, значение метки которой совпадает со значением ***K*.** После выполнения выбранной ветви происходит выход из конструкции **CASE**. Если в последовательности нет метки со значением, равным ***K*,** то управление передается внешнему оператору, следующему за конструкцией **CASE** (в случае отсутствия альтернативы ELSE; если она есть, то выполняется следующий за ней оператор, а уже затем управление передается внешнему оператору).

Запись оператора выбора

```
. . .
CASE K of
  A1 : <оп_1>;
    ...
  AN : <оп_N>
 ELSE  <оп_(N + 1)>
 end;
```

...

Любая из указанных последовательностей операторов может состоять как из единственного оператора, так и нескольких (в этом случае, как обычно, операторы, относящиеся к одной метке, должны быть заключены в операторные скобки **begin..end**).

Выражение ***K*** здесь может быть любого порядкового типа (напомним, что к таким типам относятся все целые типы, логический, литерный, перечисляемый тип, диапазонный тип, базирующийся на любом из указанных выше типов).

Задача 3. В старояпонском календаре был принят двенадцатилетний цикл. Годы внутри цикла носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки и свиньи. Написать программу, которая позволяет ввести номер года и печатает его название по старояпонскому календарю. Справка: 1996 г. — год крысы — начало очередного цикла.

Поскольку цикл является двенадцатилетним, поставим название года в соответствие остатку от деления номера этого года на 12.

```
Program Goroskop;
Var Year : Integer;
Begin
 Write('Введите год '); ReadLn(Year);
   CASE Year MOD 12 OF
    0 : WriteLn('Год Обезьяны');
    1 : WriteLn('Год Петуха');
    2 : WriteLn('Год Собаки');
    3 : WriteLn('Год Свиньи');
    4 : WriteLn('Год Крысы');
    5 : WriteLn('Год Коровы');
    6 : WriteLn('Год Тигра');
    7 : WriteLn('Год Зайца');
```

```
    8 : WriteLn('Год Дракона');
    9 : WriteLn('Год Змеи');
   10 : WriteLn('Год Лошади');
   11 : WriteLn('Год Овцы')
  end;
End.
```

Задача 4. Найти наибольшее из двух действительных чисел, используя оператор выбора.

```
Program Maximum;
Var Max, X, Y : Real;
Begin
  Write('Введите два неравных числа:');
  ReadLn(X, Y);
  Case X > Y Of
  TRUE  : Max := X;
  FALSE : Max := Y
  end;
  WriteLn('Максимальное из двух есть ', Max : 12 : 6)
End.
```

Задача 5. Преобразовать символ, если он является строчной русской буквой, в заглавную букву.

Так как в альтернативной системе кодировки ASCII строчные русские буквы идут не подряд, а с некоторым разрывом, то в данном случае, в зависимости от того, в какую часть таблицы попадает введенная буква, используется та или иная формула. Если введённый символ не является строчной русской буквой, он выводится без изменения.

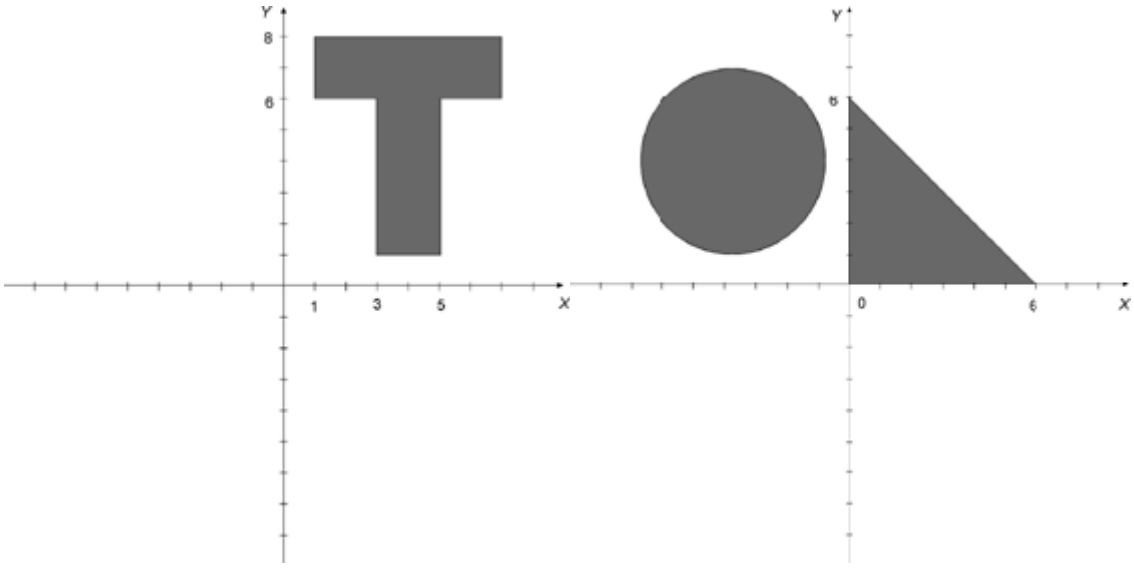
```
Program UpCase;
Var C : Char;
Begin
   Write('Введите символ:');
```

```
    ReadLn(C);
    Case C Of
    'а'..'п' : C := Chr(Ord(C) - 32);
    'р'..'я' : C := Chr(Ord(C) - 80)
    End;
    WriteLn(C);
End.
```

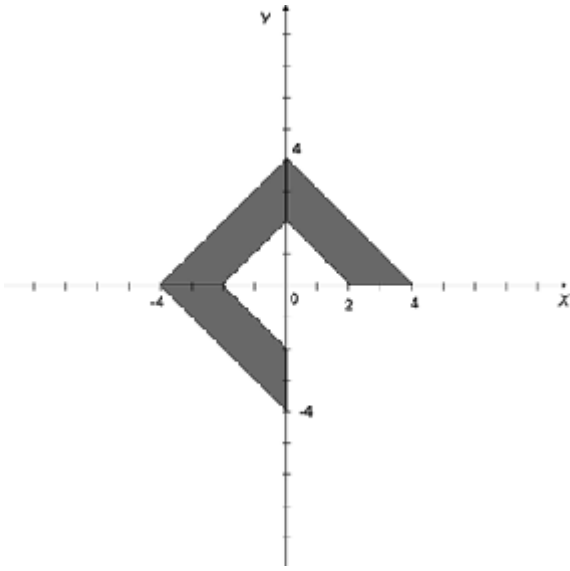
Как видно из примера, в качестве метки может выступать не только отдельное значение, но и диапазон значений. Кроме того, в качестве метки может выступать перечень значений выражения (значения перечисляются через запятую).

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

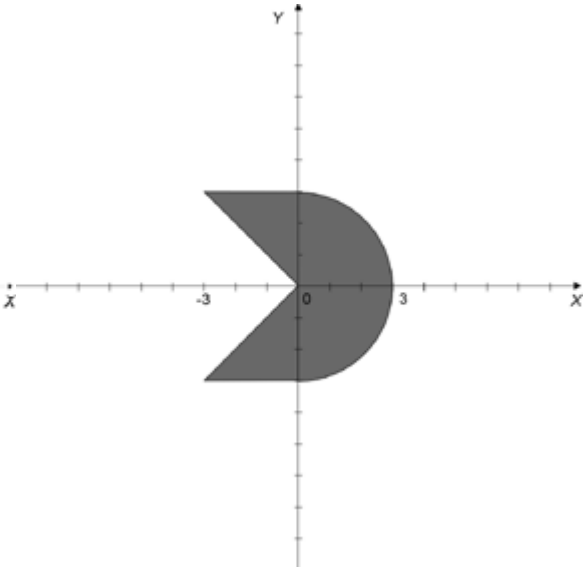
1. **Составить программу, печатающую значение TRUE, если точка с координатами (*x*, *y*) принадлежит закрашенной области, и FALSE в противном случае.**



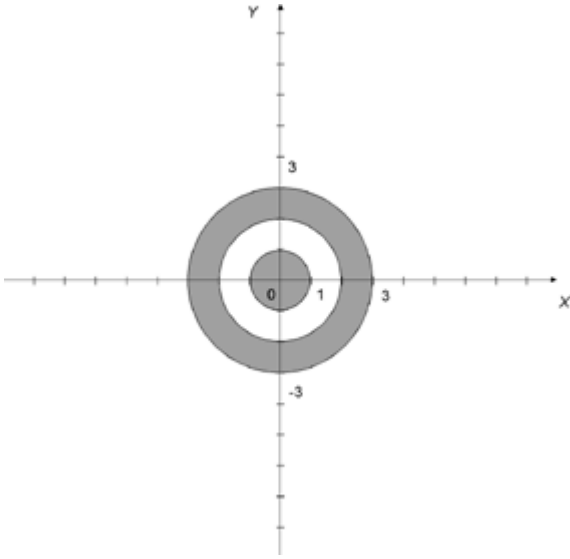
1.1. 1.2.



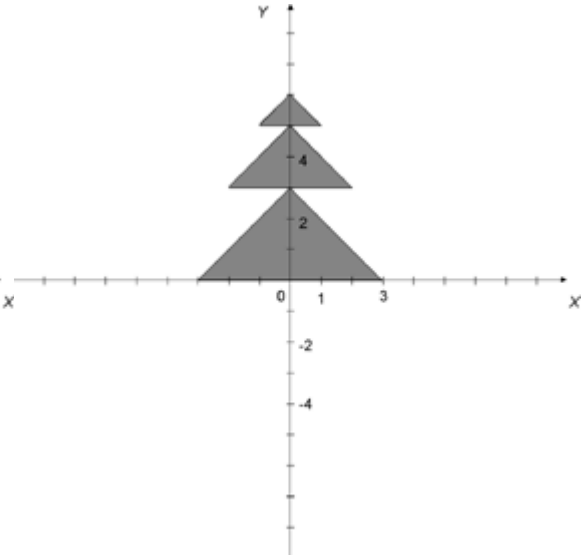
1.3.



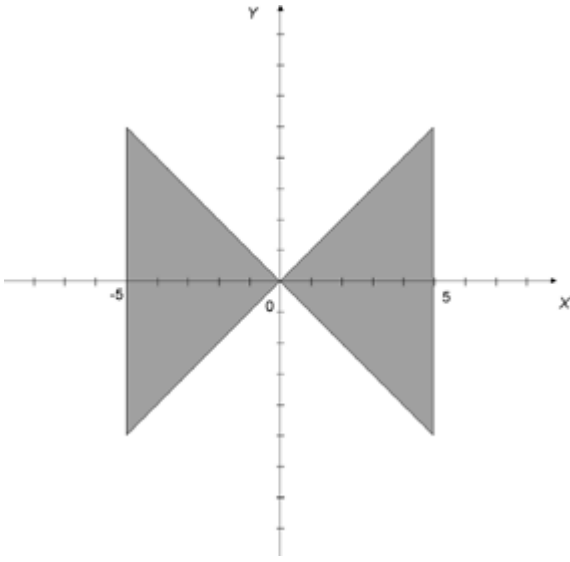
1.4.



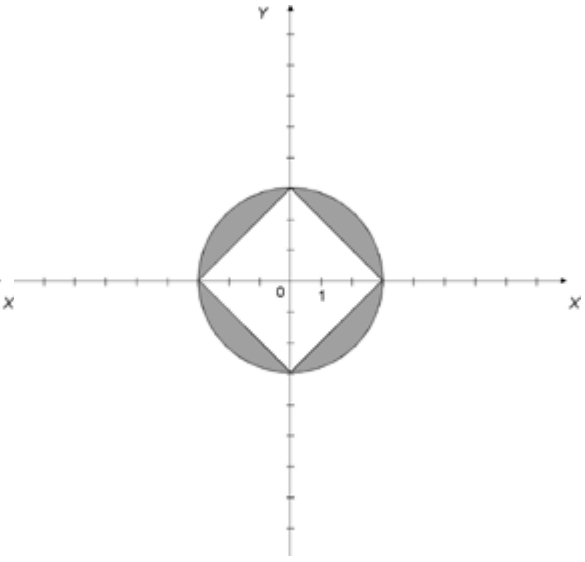
1.5.



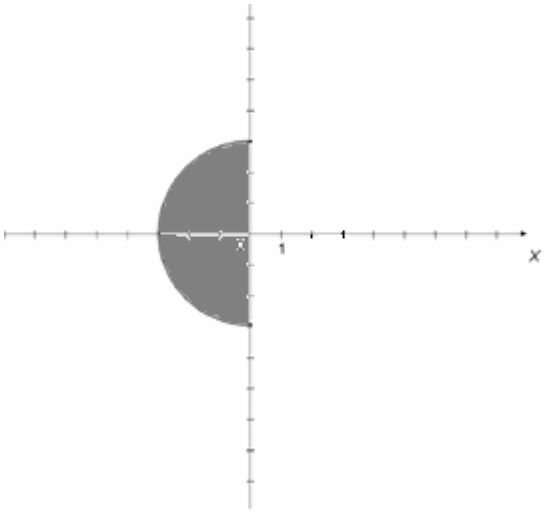
1.6.



1.7.



1.8.



Пример: Выражение
(X <= 0) and (Sqr(X) + Sqr(Y) <= 3) обозначает все точки, лежащие внутри полукруга с радиусом, равным трем и центром, совпадающим с началом координат, расположенного во второй и третьей четвертях.

2. **Составить высказывание, содержащее переменные, которое в зависимости от их значений принимает значение TRUE или FALSE. Написать программу с соответствующим логическим выражением.**

2.1. Сумма двух последних цифр заданного трёхзначного числа *N* меньше заданного *K*, а первая цифра *N* больше 5.

2.2. Сумма двух первых цифр заданного четырёхзначного числа *N* равна произведению двух последних.

2.3. Заданы три положительных числа *A*, *B*, *C*. Эти числа являются сторонами равнобедренного треугольника.

2.4. Дробь *A* / *B* является правильной.

2.5. В двузначном натуральном числе *n* первая цифра меньше второй.

2.6. Сумма цифр четырехзначного натурального числа является однозначным числом.

2.7. Число *d* является корнем уравнения $ax^2 + bx + c = 0$ или уравнения $mx + n = 0$.

2.8. Данное натуральное число *N* кратно *K*, но не кратно *L*.

Пример: *X* — отрицательное целое число, делящееся на 3 нацело.

(X < 0) And (X Mod 3 = 0)

3. Написать программу со структурой ветвления.

3.1. Написать программу, которая вычисляет частное от деления двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю) выдавать сообщение об ошибке. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление частного.
Введите в одной строке делимое и делитель,
затем нажмите <ENTER>.
-> 12  0

Вы ошиблись. Делитель не должен быть равен нулю.
```

3.2. Написать программу вычисления площади кольца. Программа должна проверять правильность исходных данных. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление площади кольца.
Введите исходные данные:
Радиус кольца (см) ->   3.5
Радиус отверстия (см)  ->  7

Ошибка! Радиус отверстия не может быть больше радиуса кольца.
```

3.3. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 10% предоставляется в том случае, если сумма покупки больше 1000 руб. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости покупки с учетом скидки.
Введите сумму покупки и нажмите <ENTER>.
-> 1200

Вам предоставляется скидка 10%.
Сумма покупки с учетом скидки: 1080.00 руб.
```

3.4. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 3% предоставляется в том случае, если сумма покупки больше 500 руб., в 5% - если сумма больше 1000 руб. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости покупки с учетом скидки.
Введите сумму покупки и нажмите <ENTER>.
-> 640

Вам предоставляется скидка 3%.
Сумма покупки с учетом скидки: 620.80 руб.
```

3.5. Написать программу, которая проверяет, является ли четным введенное пользователем целое число. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Введите целое число и нажмите <ENTER>.
-> 23
```

```
Число 23 – нечетное.
```

3.6. Написать программу, которая проверяет, делится ли на 3 целое число, введенное с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Введите целое число и нажмите <ENTER>.
-> 451

Число 451 нацело на 3 не делится.
```

3.7. Написать программу определения стоимости разговора по телефону с учетом скидки 20%, предоставляемой по субботам и воскресеньям. Ниже представлен рекомендуемый вид экрана программы во время ее работы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Вычисление стоимости разговора по телефону.
Введите исходные данные:
Длительность разговора (целое количество минут) -> 3
День недели (1 – понедельник, … , 7 – воскресенье) -> 6

Предоставляется скидка 20%.
Стоимость разговора: 5.52 руб.
```

3.8. Написать программу, которая вычисляет оптимальный вес пользователя, сравнивает его с реальным и выдает рекомендации о необходимости поправиться или похудеть. Оптимальный вес вычисляется по формуле: рост (в сантиметрах)-100. Ниже представлен рекомендуемый

вид экрана программы во время ее работы (данные, введенные пользователем, выделены полужирным шрифтом).

```
Введите в одной строке через пробел
рост (см) и вес (кг), затем нажмите <ENTER>.
-> 170 68

Вам надо поправиться на 2.00 кг.
```

4.Написать программу с использованием оператора CASE .. OF

4.1.Написать программу, которая запрашивает у пользователя номер месяца и выводит соответствующее название времени года. В случае, если пользователь укажет недопустимое число, программа должна вывести сообщение: «Ошибка ввода данных». Ниже представлен рекомендуемый вид экрана программы во время ее работы.

```
Введите номер месяца (число от 1 до 12) и нажмите
<ENTER>.
-> 11

Время года – зима.
```

4.2.Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: «Рабочий день», «Суббота» или «Воскресенье».

4.3.Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 999), обозначающего денежную единицу, дописывает слово «рубль» в правильной форме. Например, 12 рублей, 21 рубль и т.д.

4.4.Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 99), обозначающего денежную единицу, дописывает слово «копейка» в правильной форме. Например, 5 копеек, 41 копейка и т.д.

4.5.Написать программу, которая вычисляет дату следующего дня. Ниже представлен рекомендуемый вид экрана программы во время ее работы.

```
Введите цифрами сегодняшнюю дату (число месяц год)
->  31 12 1999

Последний день месяца!
С наступающим Новым Годом!
Завтра 01.01.2000
```

4.6.Написать программу, которая в зависимости от символа специализации группы выдает сообщение – «Привет, автоматчик!», «Привет, экономист!», «Привет, студент неизвестной специальности!».

4.7.Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 24), обозначающего временную единицу, дописывает слово «час» в правильной форме. Например, 5 часов, 1 час и т.д.

4.8.Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 60), обозначающего временную единицу, дописывает слово «минута» в правильной форме. Например, 5 минут, 1 минута и т.д.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 3
ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА GOTO

Существует оператор, часто используемый в совокупности с конструкций **if...then...else**. Это оператор **goto**. С его помощью программу можно сделать по-настоящему интерактивной, то есть работающей в зависимости от действий пользователя, не всегда заканчивающей работу после того, как она выполнит какие-нибудь действия.

Обычно операторы в программе выполняются в том порядке, в каком они записаны. Оператор перехода прерывает естественный порядок выполнения программы и указывает, что дальнейшее выполнение должно продолжаться, начиная с оператора, помеченного меткой, указанной в операторе перехода. Пример записи оператора перехода: **goto 218;**

Команда **goto** передает управление оператору, помеченному меткой. До того, как их использовать, метки нужно описать, - сообщить Паскалю об их наличии. Описание меток происходит в разделе описания программы.

Раздел с метками называется **label** и оформляется следующим образом:

```
Program UseGOTO;
  label  First;
  var  A,B: Integer;
  Begin
  First:  Write('Введите A: ');
            Readln(A);
            Write('Введите B: ');
            Readln(B);
            If A > B Then goto First;
          Readln;
End.
```

Этот пример хорошо демонстрирует использование оператора **goto**. Посмотрите внимательно на программу. Что она делает? Запрашивает два числа и если 1-е больше, чем 2-е, то повторяется сначала. Необходимые комментарии:

1. **label First;**

   Это и есть раздел описания меток. Служебное слово **label** озаглавливает этот раздел, после него идут имена меток. Если меток несколько, то они перечисляются через запятую.

2. **First:**

   Так устанавливается метка в программе. Обратите внимание на синтаксис - после имени метки ставиться двоеточие - **":"**.

3. **If** A > B **Then goto** First;

   А это и есть переход при выполнении условия. Заметьте, в конструкции **if...then...else** отсутствует слово **else**, оно нам не нужно, так как мы не делаем ничего при невыполнении условия.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ

С помощью оператора **goto** написать программу, реализующую следующий алгоритм.

1. Вводим два числа;
2. Складываем их и выводим сумму на экран;
3. Спрашиваем, повторить ли действие?
4. Если ответ утвердительный, переходим к пункту 1 данного алгоритма.
5. Завершаем программу.

В этом примере программа может выполняться бесконечно, пока ответ пользователя будет утвердительным.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 4
ЦИКЛИЧЕСКАЯ КОНСТРУКЦИЯ FOR .. TO .. DO

В Паскале есть и другие конструкции, позволяющие совершенствовать и усложнять тексты программ, а следовательно, расширять возможности программирования. К таким конструкциям относятся **циклы.** Циклы - это неотъемлемая часть программы. Рассмотрим, что такое циклы и когда они могут потребоваться.

Начнем с примера. Давайте представим, что нам потребовалось написать программу, которая будет выводить 10 раз число, введенное пользователем.

Решение программы очевидно:

```
1.Мы читаем в переменную ее значение.
2.Десять раз выводим ее на экран.
3.Завершаем программу.
```

Все вроде ясно. Читать строку мы будем процедурой **Readln**, как завершить программу тоже не вопрос. А вот как вывести строку 10 раз? Можно десять раз написать процедуру **Writeln**. Но это очень громоздко и нерационально. Здесь желательно организовать программу так, чтобы та ее часть, которая выводит 10 раз переменную, повторялась сама. Для этого и используются циклы. Они позволяют "зацикливать программу", то есть заставляют ее повторяться несколько раз - столько, сколько нужно пользователю.

Циклических конструкций несколько – с известным и неизвестным числом повторений.

Цикл **FOR** относиться к разряду тех, которые заставляют повторяться программу определенное количество раз.

ЦИКЛ FOR..TO..DO

Чтобы продемонстрировать, как работает этот цикл, давайте напишем программу, приведенную выше:

```
Program Example;
  var  I: Byte; V: Integer;
  Begin
  Write('Введите целое число: ');
  Readln(V);
  For I := 1 to 10 do
  Writeln(V);
Readln;
End.
```

Рассмотрим синтаксис программы:

1. Цикл начинается, словом **FOR**.

2. После него идет присваивание переменной – *параметру цикла* начального значения. Именно начиная с этого значения и будет происходить отсчет.

3. Далее идет слово **TO**.

4. После этого указывается конечное значение *параметра цикла*. До этого значения будет производиться отсчет.

5. В конце заголовка цикла ставиться слово **do**.

6. После этого идут все действия, которые должны быть зациклены. Здесь действует тоже правило, что и в конструкции **if...then...else** - про вложенность. То есть если после слова **do** должно быть несколько действий, а не одно, то все они заключаются в конструкцию **begin ... end;** Помните об этом, иначе циклы не будут правильно работать -

выполняться будет только первое действие, а остальные будут затронуты только после выполнения цикла.

Необходимо написать программу из 10 шагов, которая будет увеличивать значение введенной переменной на значение параметра цикла, распечатывая при этом каждый свой шаг, т.е. ее вывод должен быть таким (для примера возьмем, что мы ввели число 7):

```
A+1 = 8;
A+2 = 9;
A+3 = 10;
A+4 = 11;
.............
A+10 = 17;
```

Текст программы:

```
Program Example;
  var
  A: Integer;
  I: Byte;
  Begin
  Write('Введите число: ');
  Readln(A);
  For I := 1 to 10 do
  Writeln(A,' + ',I,' = ',A+I);
Readln;
End.
```

Заметьте также, что в цикле используется тип **Byte**. Дело в том, что **Byte** занимает меньше памяти, чем **Integer** или **Word**, тем самым программа становиться оптимизированнее. Мы ведь заранее знаем, что в цикле значение

не перевалит через 10, верно? Зачем же использовать типы, способные хранить значения гораздо большие?

Однако, иногда необходимо, чтобы переменная в цикле уменьшалась. Например, модифицированный вариант предыдущей программы - уменьшать на значение параметра цикла введенную переменную. Как же поступить? Именно для этих целей и существует расширение синтаксиса цикла **FOR**, которое позволяет уменьшать значения, то есть задавать "от большего к меньшему".

Для того, чтобы использовать эту возможность, необходимо заменить служебное слово **to** словом **downto**, то есть оформление станет таким:

For I := 10 downto 1 do

Давайте теперь модифицируем предыдущую программу, только уменьшать будем с 10 до 1, а не с 1 до 10, как мы делали ранее. Сделаем еще одно, чтобы усовершенствовать программу: так как мы будет вычитать сразу 10 из введенного числа, не хотелось бы, чтобы оно было меньше нуля. Итак, мы будем проверять введенное число на «больше/меньше нуля», после чего вычитать из него числа от 10 до 1. Пример программы:

```
Program Example;
  label L1;
  var A: Integer;
            I: Byte;
  Begin
  L1:  Write('Введите число: ');
  Readln(A);
  if A <= 10 then goto L1;
  For I := 10 downto 1 do
  Writeln(A,' - ',I,' = ',A-I);
```

```
Readln;
End.
```

Рассмотрим несколько задач.

Задача 1 Написать программу, выводящую в столбик десять строк, в каждой печатая цифры от **0** до **9**, то есть в таком виде:

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

.....

0 1 2 3 4 5 6 7 8 9

Чтобы вывести столбик цифр, больше всего подойдут циклы, причем использовать мы будем известный нам цикл **FOR**. Мы его запустим 10 раз, каждый раз будем выводить 10 цифр, которые также будут выводиться циклом, причем после этого нужно будет перенести курсор на новую строку, чтобы получался столбик. Сразу же решаем, какой тип мы будем использовать в циклах. Подойдет тип **Byte**, не так ли? Ведь значения у нас не будут переходить через 255 (границу Byte). Итак, текст программы:

```
Program N1;
  var
  I,J: Byte;
  Begin
  For I := 1 to 10 do
  begin
  For J := 0 to 9 do
  Write(J, ' ');
  Writeln;
  end;
```

```
Readln;
End.
```

Заметьте, здесь можно вместо второго цикла, печатающего строку цифр, можно использовать процедуру Write:

Write('0 1 2 3 4 5 6 7 8 9 ');

Но, тем не менее, это не было сделано для того, чтобы продемонстрировать вложенность циклов. Следующий пример будет являться усовершенствованным этой задачи.

Задача 2 Написать программу, распечатывающую таблицу умножения. (По типу известной всем Таблицы Пифагора). Программа должна выводить следующий экран:

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

.....

10 20 30 40 50 60 70 80 90 100

Будем использовать два цикла - один будет вложен в другой. Только нужно соотнести значения циклов так, чтобы на пересечении строки и столбца было произведение соответствующих чисел. Рассмотрим крайне простой и оптимизированный алгоритм, который распечатывает таблицу умножения.

Что представляет собой таблица умножения? Это произведение строки на столбец. Зная это, мы решаем поставленную перед нами задачу: первый цикл у нас будет считать строки, так ведь? (Вспомните предыдущую программу), а второй (вложенный цикл) считает символы в строке, или столбцы. То есть, чтобы получить произведения строки на столбец, нужно просто умножать значения циклов друг на друга!

```
Program N2;
  var
```

```
  I,J: Byte;
  begin
  For I := 1 to 10 do
  begin
  For J := 1 to 10 do
  If I*J > 10 then Write(I*J, ' ')
  else Write(I*J, '  ');
  Writeln;
  end;
Readln;
end.
```

Видите, мы всего лишь умножаем индексы циклов друг на друга, вот и все! Теперь внимательно смотрите на обе процедуры **Write**.

В первой из них один пробел - **Write(I*J,' . ');**

Во второй два - **Write(I*J,' . . ');**

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Написать программу, которая выводит таблицу квадратов и кубов первых десяти целых положительных чисел. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица квадратов и кубов
------------------------------
Число !  Квадрат !   Куб
------------------------------
  1    !    1    !    1
  2    !    4    !    4
  3    !    9    !   27
```

```
  4     !    16     !    64
. . .
-------------------------------
```

2. Написать программу, которая выводит таблицу квадратов и кубов первых десяти целых положительных нечетных чисел. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица квадратов и кубов
нечетных чисел
-------------------------------
Число !  Квадрат !   Куб
-------------------------------
  1     !     1     !     1
  3     !     9     !    27
  5     !    25     !   125
. . .
-------------------------------
```

3. Написать программу, которая выводит таблицу пяти шестизначных четных чисел, делящихся без остатка на сумму своих цифр. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица шестизначных чисел
-----------------------------------------
Число   !  Сумма цифр !   Частное
-----------------------------------------
 xxxxxx !      zzz          !    yyy
. . .
-----------------------------------------
```

4. Написать программу, которая выводит таблицу пяти трехзначных чисел, делящихся без остатка на произведение своих цифр. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица трехзначных чисел
----------------------------------------
Число   !  Произведение !    Частное
----------------------------------------
 xxx   !          zzz         !    y
. . .
----------------------------------------
```

5. Написать программу, которая выводит таблицу пяти шестизначных «счастливых» чисел (сумма первых трех чисел равна сумме трех последних). Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица шестизначных «счастливых» чисел
------------------------------
N п/п !  Число   !  Сумма !
------------------------------
    1  !  xxxxxx  !    zzz    !
. . .
------------------------------
```

6. Написать программу, которая выводит таблицу десяти длин и площадей окружностей радиусом 1..10 . Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица длин и площадей окружности
----------------------------------------
Радиус !  Длина окр  !  Площадь !
```

```
-------------------------------------
   1   !     6,28     !     3,14   !
. . .
-------------------------------------
```

7. Написать программу, которая выводит таблицу десяти пятизначных симметричных нечетных чисел (например, 34543 или 70507) . Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица пятизначных симметричных чисел
------------------
N п/п !  Число   !
------------------
   1  !  xxxxx   !
. . .
--------------------
```

8. Написать программу, которая выводит таблицу шести целых степеней числа ***Пи***. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица степеней числа Пи
---------------------------
Показатель !  Степень  !
---------------------------
   1        !     3,14  !
. . .
---------------------------
```

9. Написать программу, которая вычисляет сумму первых ***n*** целых положительных четных чисел. Количество суммируемых чисел

вводится с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление суммы четных чисел.
Введите количество чисел и нажмите <Enter>
-> 12
Сумма первых 12 четных чисел равна 156.
```

10. Написать программу, которая вычисляет сумму первых ***n*** целых положительных нечетных чисел. Количество суммируемых чисел вводится с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление суммы нечетных чисел.
Введите количество чисел и нажмите <Enter>
-> 12
Сумма первых 12 нечетных чисел равна 144.
```

11. Написать программу, которая вычисляет факториал целого положительного числа ***n*** (тип Byte) . Число ***n*** вводится с клавиатуры. Осуществить проверку на превышение MaxInt. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление факториала.
Ведите число, факториал которого необходимо вычислить
-> 5
Факториал числа 5 равен 120.
```

12. Написать программу, которая вводит с клавиатуры ***5*** дробных чисел и вычисляет их среднее арифметическое. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление среднего арифметического последовательности дробных чисел. После ввода каждого числа нажмите <Enter>
-> 5.4
-> 7.8
-> 3.0
-> 1.5
-> 2.3

Среднее арифметическое введенной последовательности: 4.0
Для завершения работы нажмите <Enter>
```

13. Написать программу, которая вводит с клавиатуры ***5*** дробных чисел и после ввода каждого числа выводит среднее арифметическое полученной части последовательности. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Обработка последовательности дробных чисел.
После ввода каждого числа нажмите <Enter>
-> 12.3
Введено чисел: 1 Сумма: 12.30  Сред.арифметическое: 12.30
-> 15
Введено чисел: 2 Сумма: 27.30  Сред.арифметическое: 13.65
-> 10
Введено чисел: 3 Сумма: 37.30  Сред.арифметическое: 12.43
-> 5.6
Введено чисел: 4 Сумма: 42.90  Сред.арифметическое: 10.73
-> 11.5
Введено чисел: 5 Сумма: 54.40  Сред.арифметическое: 10.88

Для завершения работы нажмите <Enter>
```

14. Написать программу, которая вычисляет среднее арифметическое последовательности дробных чисел, вводимых с клавиатуры. После того, как будет введено последнее число, программа должна вывести минимальное и максимальное число последовательности. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Обработка последовательности дробных чисел.
Введите количество чисел последовательности -> 5
Вводите последовательность. После ввода каждого числа
нажмите <Enter>
-> 5.4
-> 7.8
-> 3.0
-> 1.5
-> 2.3

Количество чисел: 5
Среднее арифметическое: 4.00
Минимальное число: 1.5
Максимальное число: 7.8
Для завершения работы нажмите <Enter>
```

15. Написать программу, которая выводит минимальное и максимальное число последовательности дробных чисел, вводимых с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Обработка последовательности дробных чисел.
Введите количество чисел последовательности -> 5
Вводите последовательность. После ввода каждого числа нажмите <Enter>
```

```
-> 5.4
-> 7.8
-> 3.0
-> 1.5
-> 2.3

Количество чисел: 5
Минимальное число: 1.5
Максимальное число: 7.8
Для завершения работы нажмите <Enter>
```

16. Напишите программу, которая выводит на экран квадрат Пифагора – таблицу умножения. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Таблица умножения

      1    2    3    4    5    6    7    8    9   10
   |-----------------------------------------------
 1|  1    2    3    4    5    6    7    8    9   10
 2|  2    4    6    8   10   12   14   16   18   20
 3|  3    6    9   12   15   18   21   24   27   30
…
```

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 5
ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ
С НЕИЗВЕСТНЫМ ЧИСЛОМ ПОВТОРЕНИЙ
REPEAT … UNTIL, WHILE … DO

Цикл **Repeat … Until** весьма часто используется в программировании. Он довольно удобен для организации больших структур данных, оформления целых блоков программы. Отличительная особенность цикла - это проверка подлинности выполнения не в начале, как у других циклов, а в конце. Поэтому такой цикл называется циклом с постусловием.

Проверка может быть на что угодно:

- На подлинность булевской переменной;
- На наличие какого-нибудь условия, например, была введена буква "q", означающая выход из программы;
- На значение какой-нибудь переменной (а не больше ли она пяти?)
- и т.д...

Для того, чтобы озаглавить цикл **REPEAT . . UNTIL** (кстати, по английски **repeat** - повторять, а **until** - пока, до того как) необходимо в любом месте программы вставить служебное слово **REPEAT**. Именно одно слово, не как во всех других циклах. Завершается цикл служебным словом **UNTIL** с последующей проверкой какого-нибудь условия. Пример:

```
Program N3;
  var
  S: String;
  Exit: Boolean;
  Begin
  REPEAT
  Write('Введите строку (end - выход): ');
```

```
  Readln(S);
  if S = 'end' then Exit := true;
  UNTIL Exit;
  Write('Конец программы...');
Readln;
End.
```

В этом примере проверка происходит на подлинность булевской переменной, которая устанавливается если введена строка 'end'.

Обратите внимание, что после слова **REPEAT** не ставится точка с запятой. Дело в том, что **REPEAT** озаглавливает целый блок, поэтому знак **";"** отсутствует. Во-вторых, операторы внутри конструкции **REPEAT . . UNTIL** не выделяются дополнительными **begin..end**. Хотя операторов и несколько, этого не происходит.

Ну а теперь еще пример, на этот раз проверяем числовую переменную:

```
Program N4;
  var
  A: Integer;
  Begin
  REPEAT
  Write('Введите число (0 - выход): ');
  Readln(A);
  UNTIL A=0;
  Write('Конец программы...');
Readln;
End.
```

Из программы видно, что она будет повторяться пока не будет введен нуль.

Цикл **While** имеет свои специфические особенности использования и выполнения.

Работая с циклом **FOR,** вы заметили, что программа повторяется всегда фиксированное количество раз - пусть даже это количество и задается в ходе выполнения программы, но изменять его в ходе выполнения цикла нельзя. Если сказано прокрутить цикл десять раз, десять раз он и прокрутиться. Наводит на мысль, что цикл **FOR** использовать удобно при подходящей ситуации - например, каких-нибудь математических расчетах или внутреннем выполнении действий. А вот если нам понадобилось внешняя работа с данными, циклически оформленными? Это, например, может быть программа ввода строки, которая будет читать строку, пока та не будет содержать слова "end". Выполнение этого цикла несколько специфичное - он работает до возникновения определенных условий (со стороны внешних данных).

В заголовке этого цикла стоит не диапазон значений, а собственная процедура проверки - вроде известной нам **if...then...else**.

Напишем программу, которая и будет выполнять чтение строки до того момента, пока она не будет строкой "end". Смотрите программу:

```
Program N1;
  var
  S: String;
  Begin
  While S <> 'end' do
  Readln(S);
  Write('Вот и все! Вы ввели end!');
Readln;
End.
```

Программа читает строки до того момента, пока введенная строка не будет равна "end"? Внимательно посмотрите на программу. Вместо типичного диапазона значений цикла (как **FOR** - повторять от сих до сих) стоит процедура проверки, то есть "повторять пока". Кстати, **while** переводиться с английского как **"пока".** Тело цикла может не выполниться ни одного раза, если результат логического условия будет **false. Такой цикл называется циклом с предусловием.** Теперь необходимые комментарии к изученному циклу.

4. Для того, чтобы оформить цикл **while** предназначено служебное слово **while.**
5. После указания этого слова идет логический диапазон цикла, то есть нужно написать, при каких условиях цикл закончиться. Это реализуется посредством известных нам знаков сравнения и их взаимоотношения с переменными.
6. После указания условия ставиться служебное слово **do.**
7. Теперь идет само тело цикла. Здесь помните, что если в теле цикла содержится один оператор (как в первом примере), то он указывается без дополнительных выделений. Если же идет несколько операторов, то они все заключаются в конструкцию **begin-end.**

Итак, мы решили написать калькулятор. Для его работы нужны два числа и знак действия - все это мы будем вводить по отдельности, так как пока не умеем разбивать строки на числа. Общий алгоритм таков:

6. Ввести A;
7. Ввести B;
8. Ввести знак действия;

9. В зависимости от того, что это за знак, выполнить действие:
    1. **Это "+"?**

       -> C = A + B;
    2. **Это "-"?**

       -> C = A - B;
    3. **Это "*"?**

       -> C = A * B;
10. Вывести результат;
11. Спросить - сначала?
12. Если ответ утвердительный, то начать все сначала (переход к пункту 1);
13. Конец нашей программки.

Вот исходный текст, реализующий этот алгоритм:

```
Program Simple_Calculator;
  var
  A,B,C: Integer;
  Ch, Sign: Char;
  Begin
  While UpCase(Ch) <> 'N' do
  begin
  Write('Введите A: ');
  Readln(A);
  Write('Введите B: ');
  Readln(B);
  Write('Введите знак действия: ');
  Readln(Sign);
```

```
  If Sign='+' then C := A + B;
  If Sign='-' then C := A - B;
  If Sign='*' then C := A * B;
  Writeln('Результат: ', C);
  Write('Сначала? (Y/N): ');
  Readln(Ch);
  end;
  Write('Калькулятор завершает свою работу...');
Readln;
End.
```

В этой программе используется функцию **UpCase**.

Функция **UpCase** - возвращает как результат своей работы переменную-символ в верхнем регистре. В качестве параметра к функции задается переменная типа **Char**. Если эта переменная является буквой в нижнем регистре, она превратиться в большую, заглавную букву. В других случаях (например, переменная **Char** является цифрой) ничего не произойдет. Это и есть вся работа функции **UpCase**.

Напишем программу, разбивающую введенное число на разряды, то есть выделяющая в нем отдельно сотни тысяч, тысячи, сотни, десятки и единицы.

Запишем алгоритм работы программы:

4. Вводим число.
5. Пока это число больше нуля, делаем следующее:
    2.1. Получаем остаток от деления введенного числа на 10; Получившийся остаток и будет первым разрядом, т.е. единицами;
    2.2. Вычитаем получившийся остаток из имеющегося числа;

2.3. Делим получившееся число на 10 без остатка;

2.4. Выводим его на экран.

6. Спрашиваем, начать ли сначала?

7. Если да, то переходим к пункту 1.

8. Завершаем программу.

Программа:

```
Program Get_numbers;
  var
  A, B: Integer;
  Ch: Char;
  Begin
  While UpCase(Ch) <> 'N' do
  begin
  Write('Введите число: ');
  Readln(A);
  While A > 0 do
  begin
  B := A mod 10;
  Dec(A, B);
  A := A div 10;
  Writeln('Разряд: ', B);
  end;
  Write('Сначала? (Y/N): ');
  Readln(Ch);
  end;
```

```
  Write('Конец программы...');
Readln;
End.
```

Комментарии к программе.

В нашей программе главной задачей является выявление всех цифр введенного числа, или его разрядов. Это реализуется за счет уменьшения порядка числа, предварительно получая последний знак.

При получении остатка от деления числа на 10 выявляется последний, самый младший разряд, то есть та цифра в числе, которая меньше десяти. Это и есть первый разряд. Выводим его на экран, после чего уменьшаем число на это значение:

Dec(A, B);

Это эквивалентно:

A := A - B;

Вот весь ход выполнения операции получения последнего знака:

* К примеру, мы ввели число 157:

1. 157 **mod** 10 = 7; (Вот он, последний знак!)
2. 157 - 7 = 150; (А вот мы его и отрезали!)

* Если мы ввели число 1:

1. 1 **mod** 10 = 1; (Опять последний знак - он же и первый)
2. 1 - 1 = 0;
   (И опять мы его отрезаем, причем число стало равным нулю - оно как бы кончилось, однозначное ведь).

Итак, после отсечения последней цифры нам нужно уменьшить это число в десять раз, то есть сдвинуть на один знак вправо, чтобы получить возможность выполнить все действия с самого начала и выявить следующий разряд. Для этого мы выполняем деление на 10 без остатка.

120 **div** 10 = 12;

10020 **div** 10 = 1002;

10 **div** 10 = 1;

Видите, число подвигается на один разряд? Ну а теперь вспомните наши манипуляции с операцией **mod** - отсечение последнего знака. Что же получиться, если после того, как мы сдвинули число опять выполнить вышеописанные действия? Верно, мы снова получим остаток от деления или последний разряд.

Ну а теперь пустим эти две операции по кругу, организуем цикл. Причем цикл будет "пока число больше нуля", то есть пока оно у нас не кончиться. Сюда хорошо вписывается цикл **While**.

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

Написать каждое задание с помощью двух циклических конструкций – с предусловием и с постусловием.

17. Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление среднего арифметического последовательности положительных чисел
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> 45
-> 23
-> 15
-> 0

Введено чисел: 3
```

```
Сумма чисел: 83
Среднее арифметическое: 27.67
```

18. Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Определение максимального числа последовательности положительных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> 56
-> 75
-> 43
-> 0

Максимальное число: 75.
```

19. Написать программу, которая проверяет, является ли целое число, введенное пользователем простым. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Введите целое число и нажмите <Enter>
-> 45

45 – не простое число.
```

20. Написать программу, которая вычисляет наибольший общий делитель двух целых чисел.

21. Написать программу, которая вводит с клавиатуры ***n*** дробных чисел и вычисляет их среднее арифметическое. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление среднего арифметического последовательности дробных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> 5.4
-> 7.8
-> 3.0
-> 1.5
-> 2.3
-> 0

Количество чисел: 5
Среднее арифметическое введенной последовательности: 4.0
Для завершения работы нажмите <Enter>
```

22. Написать программу, которая вычисляет среднее арифметическое последовательности дробных чисел, вводимых с клавиатуры. После того, как будет введено последнее число, программа должна вывести минимальное и максимальное число последовательности. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Обработка последовательности дробных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.
-> 5.4
-> 7.8
-> 3.0
```

-> **1.5**
-> **2.3**
-> **0**

Количество чисел: 5
Среднее арифметическое: 4.00
Минимальное число: 1.5
Максимальное число: 7.8
Для завершения работы нажмите <Enter>

23. Написать программу, которая выводит минимальное и максимальное число последовательности дробных чисел, вводимых с клавиатуры. Ниже представлен рекомендуемый вид экрана во время работы программы.

Обработка последовательности дробных чисел.
Вводите после стрелки числа. Для завершения ввода введите ноль.

-> **5.4**
-> **7.8**
-> **3.0**
-> **1.5**
-> **2.3**
-> **0**

Количество чисел: 5
Минимальное число: 1.5
Максимальное число: 7.8

Для завершения работы нажмите <Enter>

24. Написать программу, которая вычисляет сумму первых ***n*** целых положительных нечетных чисел. Ниже представлен рекомендуемый вид экрана во время работы программы.

```
Вычисление суммы нечетных чисел.
Введите количество чисел и нажмите <Enter>
-> 12
Сумма первых 12 нечетных чисел равна 156.
```

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 6
РАСЧЕТ КОНЕЧНОЙ И БЕСКОНЕЧНОЙ СУММЫ

Для организации накопления суммы или значения очередного члена некоторого ряда удобно использовать метод итераций, то есть пошаговое приближение к поставленной цели. В задачах на использование итерационных алгоритмов реализуется тот или иной циклический процесс, который выполняется либо за заранее известное число шагов, либо до достижения некоторого условия. Часто в задачах для вычисления очередного члена ряда удобно рекуррентно использовать предыдущее слагаемое, а не использовать дополнительный цикл. В итерационных алгоритмах заданная погрешность используется для проверки модуля разности найденного приближенного и точного значений, однако если последнее неизвестно, допустимо оценивать разность между соседними итерациями (приближениями).

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

Расчет конечных сумм

В приводимых задачах необходимо составить программу расчета конечной суммы и сравнения полученного результата с контрольным значением. Число членов суммы вводится с клавиатуры с защитой от возможного неверного ввода данных.

№	вид суммы	контрольное значение
1	$1+2+3+4+...+N$	$\frac{N(N+1)}{2}$
2	$1+3+5+7+...+(2N-1)$	N^2
3	$2+4+6+8+..+2N$	$N(N+1)$

№	вид суммы	контрольное значение
4	$1^2+2^2+3^2+4^2+...+N^2$	$\frac{N(N+1)(2N+1)}{6}$
5	$1^2+3^2+5^2+...+(2N-1)^2$	$\frac{N(4N^2-1)}{3}$
6	$1^3+2^3+3^3+4^3+...+N^3$	$\frac{N^2(N+1)^2}{4}$
7	$1^3+3^3+5^3+...+(2N-1)^3$	$N^2(2N^2-1)$
8	$1^4+2^4+3^4+4^4+...+N^4$	$\frac{(N^2+N)(2N+1)(3N^2+3N-1)}{30}$

Расчет бесконечных сумм

В приводимых задачах необходимо составить программу расчета бесконечной суммы обратных степеней числового ряда. Суммирование проводить, пока очередной член ряда по модулю не станет меньше заданной точности ε. Результат сравнить с точным значением S_T., а погрешность сопоставить с величиной ε.

№	Вид суммы	N	Вид ряда	S_T	ε
1	$\sum_{i=1}^{\infty} i^{-N}$	2	$1+\frac{1}{2^2}+\frac{1}{3^2}+...$	$\frac{\pi^2}{6}$	10^{-4}
2		4	$1+\frac{1}{2^4}+\frac{1}{3^4}+...$	$\frac{\pi^4}{90}$	10^{-6}
3	$\sum_{i=1}^{\infty}(-1)^{i-1} i^{-N}$	2	$1-\frac{1}{2^2}+\frac{1}{3^2}-...$	$\frac{\pi^2}{12}$	10^{-5}
4		4	$1-\frac{1}{2^4}+\frac{1}{3^4}-...$	$\frac{7\pi^4}{720}$	10^{-7}
5	$\sum_{i=1}^{\infty}(2i-1)^{-N}$	2	$1+\frac{1}{3^2}+\frac{1}{5^2}+...$	$\frac{\pi^2}{8}$	10^{-4}
6		4	$1+\frac{1}{2^4}+\frac{1}{3^4}+...$	$\frac{\pi^4}{96}$	10^{-5}

№	Вид суммы	N	Вид ряда	S_T	ε
7	$\sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{(2i-1)^N}$	2	$1-\frac{1}{3^2}+\frac{1}{5^2}-\ldots$	$\frac{\pi}{4}$	10^{-4}
8		4	$1-\frac{1}{3^4}+\frac{1}{5^4}-\ldots$	$\frac{\pi^3}{32}$	10^{-5}

ЛАБОРАТОРНАЯ РАБОТА № 7
СТРОКОВЫЙ ТИП ДАННЫХ STRING

Далее познакомимся с типом данных, который относится к числу структурированных. Это строковый тип данных (строка). Строка — это последовательность символов. Каждый символ занимает 1 байт памяти (код ASCII). Количество символов в строке называется ее *длиной*. Длина строки может находиться в диапазоне от 0 до 255. Строковые величины могут быть константами и переменными. Особенностью строки в **Turbo Pascal** является то, что с ней можно работать как с массивом символов, с одной стороны, и как с единым объектом, — с другой. За счет этого обработка строк достаточно гибка и удобна. *Строковая константа* есть последовательность символов, заключенная в апострофы. Например: 'это строковая константа', ‘272’. *Строковая переменная* описывается в разделе описания переменных следующим образом:

Var <идентификатор> : string[<максимальная длина строки>];

Например:

```
Var Name : string[20];
```

Параметр длины может и не указываться в описании. В таком случае подразумевается, что он равен максимальной величине — 255. Например:

Var slovo : string.

Строковая переменная занимает в памяти на 1 байт больше, чем указанная в описании длина. Дело в том, что один (нулевой) байт содержит значение текущей длины строки. Если строковой переменной не присвоено никакого значения, то ее текущая длина равна нулю. По мере заполнения строки символами ее текущая длина возрастает, но она не должна превышать максимальной по описанию величины.

Символы внутри строки индексируются (нумеруются) от единицы. Каждый отдельный символ идентифицируется именем строки с

индексом, заключенным в квадратные скобки. Например: **N[5], S[i], slovo[k+l].** Индекс может быть положительной константой, переменной, выражением целого типа. Значение индекса не должно выходить за границы описания. Обращение **S_string[0]** обеспечивает доступ к нулевому байту, где содержится значение текущей длины строки.

Тип **string** и стандартный тип **char** совместимы. Строки и символы могут употребляться в одних и тех же выражениях.

Строковые выражения строятся из строковых констант, переменных, функций и знаков операций. Над строковыми данными допустимы операции сцепления и операции отношения.

Если значение переменной после выполнения оператора присваивания превышает по длине максимально допустимую при описании величину, все лишние символы справа отбрасываются.

Пример:

Описание A	Выражение	Результат
A: string[6];	A:= 'Группа 1 '	'Группа'
A: string[2];	A:= 'Группа 1 '	'Гр'

Операция ***сцепления (конкатенации)*** (+) применяется для соединения нескольких строк в одну результирующую строку. Сцеплять можно как строковые константы, так и переменные.

Пример: 'Мама ' + 'мыла ' + 'раму'. В результате получится строка: 'Мама мыла раму'. Длина результирующей строки не должна превышать 255.

Операции отношения: =, <, >, <=, >=, <>. Позволяют произвести сравнение двух строк, в результате чего получается логическое значение (**true** или **false**). Операция отношения имеет приоритет более низкий, чем операция сцепления. Сравнение строк производится слева направо

до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в таблице символьной кодировки. Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки равны, если они полностью совпадают по длине и содержат одни и те же символы.

Пример:

Выражение	Результат
'True1'<'True2'	True
'Mother'>'MOTHER'	True
'Мама ' <> 'Мама'	True
'Cat'='Cat'	True

Функция **Copy(S, Pozition, N)** выделяет из строки ***S*** подстроку длиной ***N*** символов, начиная с позиции **Pozition.** Здесь ***N*** и **Pozition —** целочисленные выражения. **Пример:**

Значение S	Выражение	Результат
'Мама мыла раму'	Copy(S, 6, 4)	'мыла'
'Маша ела кашу'	Copy(S, 1, 8)	'Маша ела'

Функция **Concat(S1, S2, …, SN)** выполняет сцепление (конкатенацию) строк **S1, S2, …, SN** в одну строку.

Пример:

Выражение	Результат
Concat('Маша ', 'ела ', 'кашу')	'Маша ела кашу'

Функция **Length(S)** — определяет текущую длину строки ***S.*** Результат — значение целого типа.

Пример:

Значение S	Выражение	Результат
'test-5'	Length(S)	6
'(A+B)*C'	Length(S)	7

Функция **Pos(S1, S2)** — обнаруживает первое появление в строке **S2** подстроки **S1.** Результат — целое число, равное номеру позиции, где находится первый символ подстроки **S1.** Если в **S2** подстроки **S1** не обнаружено, то результат равен **0**.

Пример:

Значение S2	Выражение	Результат
'abcdef'	Pos('cd', S2)	3
'abcdcdef'	Pos('cd', S2)	3
'abcdef'	Pos('k', S2)	0

Функция **UpCase(Ch) –** преобразует строчную литеру в прописную. Параметр и результат имеют литерный тип. Обрабатывает только буквы латинского алфавита.

Пример:

Значение Ch	Выражение	Результат
'f'	UpCase(Ch)	'F'

Процедура **Delete(S, Poz, N)** — удаление ***N*** символов из строки ***S***, начиная с позиции **Poz.**

Пример:

Исходное значение S	Оператор	Конечное значение S
'abcdefg'	Delete(S,3,2)	'abefg'
'abcdefg'	Delete(S,2,6)	'a'

В результате выполнения процедуры уменьшается текущая длина строки в переменной ***S***.

Процедура **Insert(S1, S2, Poz)** — вставка строки **S1** в строку **S2,** начиная с позиции **Poz.**

Пример:

Исходное значение S2	Оператор	Конечное значение S2
'ЭВМ PC'	Insert('IBM-', S2, 5)	'ЭВМ IBM-PC'
'Рис. 2'	Insert('N', S2, 6)	'Рис. N 2'

Процедуры преобразования типов.

Процедура **Str(IB,St) –** преобразует числовое значение переменной целого или вещественного типа IB в переменную **St** типа **string.**

Пример:

Исходное значение IR	Оператор	Конечное значение St
1500	Str(IR:6,St)	'__1500'
4.8E+03	Str(IR:10,St)	'_4.800E+03'
4.8E+03	-Str(IR:8:4,St)	'-4800.00'
76854(тип longint)	-Str(IR:3,St)	'-7684'(позиции добавились до недостающих)

76854(тип real)	-Str(IR:3,St)	'-7.7E+04'

Процедура **Val(St,IB,Cod) –** преобразует переменную **St** типа **string** в числовое значение переменной целого или вещественного типа.. **Cod** – код ошибки, равен **0,** если преобразование выполнено, равен **N,** если преобразование не выполнено, при этом **N** – позиция неправильного символа.

Исходное значение St	Оператор	Конечное значение S	Конечное значение Cod
'1450'	Val(St,IR,Cod)	1450	0
'14.2E+02'	Val(St,IR,Cod)	1420.00	0
'14.2A+02'	Val(St,IR,Cod)	?	5

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Зашифровать введенную с клавиатуру строку заменой символов на символы с кодом, большим на 3 единицы. Провести дешифровку.
2. Зашифровать введенную с клавиатуры строку, поменяв местами первый символ со вторым, третий с четвертым и т.д.
3. Зашифровать введенную с клавиатуры строку, поменяв местами первый символ с третьим, второй с четвертым и т.д.
4. Найти и заменить определенный символ в строке, введенной с клавиатуры. Программа должна запрашивать заменяемый и заменяющий символы, а также подтверждение каждой замены символа с сообщением о номере его позиции в строке.
5. Определить и вывести на экран номера позиций и количество повторений запрашиваемого символа в строке, введенной с клавиатуры.

6. Определить количество слов в строке, введенной с клавиатуры (за слова принимать части строки, отделяющиеся друг от друга одним или несколькими пробелами).
7. Определить самое короткое и самое длинное слово во введенной строке.
8. Проверить, имеется ли в заданном тексте баланс открывающих и закрывающих скобок.
9. Дан текст, найти наибольшее количество цифр, идущих в нем подряд.
10. Дан текст, выяснить, является ли этот текст идентификатором.
11. Дано натуральное число *n.* Получить символьное представление *n* в виде последовательности цифр и пробелов, отделяющих группы по три цифры, начиная справа. Например, если *n=1753967*, то должно получиться ‘1 753 967’.
12. Дан текст, найти число таких групп букв, которые начинаются и кончаются одной и той же буквой.
13. Дан текст, определить, содержит ли он символы, отличные от букв и пробела.
14. Дан текст, выяснить, является ли этот текст десятичной записью целого числа.
15. Дан текст, заменить первый пробел текста на 1, второй – на 2 и т.д.
16. Найти все такие группы букв. в которые буква *a* входит не менее двух раз.
17. Если в данном тексте нет символа *, то оставить текст без изменения, иначе заменить символ перед * на цифру 3.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 8
“М А С С И В Ы “

Массив представляет собой структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип. Элементами массива могут быть данные любого типа. Тип элементов массива называется базовым. Число элементов массива в процессе выполнения программы не меняется. Доступ к отдельному элементу осуществляется путем индексирования элементов массива. В программах на языке программирования Турбо Паскаль элемент массива обозначается ***A[i,j].*** Для описания массивов предназначено словосочетание ***array of*** (массив из).

Форматы представления данных типа “массив”:

1. Явное описание типа:

Type

<имя типа>= ***array*** [тип индекса] ***of*** [тип компонент];

Var <идентификатор> : <имя типа>;

2. Неявное описание типа:

Var <идентификатор> : ***array*** [тип индекса] ***of*** [тип компонент];

Пример 1:

```
Type  Klass=(K1,K2,K3,K4);
      Znak=array [1..255] of char;
 Var  M1:Znak;
      M2:array [1..60] of integer;
      M3:array[1..4] of Klass;
      Name:array[1..5] of string[25];
```

Если в качестве базового типа взят другой массив, образуется многомерный массив.

Пример 2:

```
Type  Vektor=array[1..4]  of  integer;
      Massiv=array[1..4]  of  Vektor;
 Var  Matr : Massiv;
      Matriz : array [1..4,1..4]  of integer;
```

Задание 1. Ввести с клавиатуры и вывести на экран элементы матрицы размером 3x2.

```
Program Massiv;
  Var   Matriz : array [1..3,1..2]  of  integer;
        i,j : integer ;
Begin
       WriteLn('Введите элементы матрицы: ');
For i:=1 to 3 do
          For j:=1 to 2 do
              Read(Matriz[i,j]);
WriteLn('Матрица');
For i:=1 to 3 do begin
        For j:=1 to 2 do
            Write(Matriz[i,j]:5);
        WriteLn
                  end
End.
```

Задание 2. Заполнить матрицу 3x2 с помощью генератора случайных чисел числами от -25 до 24.

```
Program Massiv;
  Var
        Matriz : array [1..3,1..2]  of  integer;
        i,j : integer;
Begin
 Randomize;{Изменить    начальное    число    генератора
случайных чисел}
```

```
  WriteLn('Введите элементы матрицы: ');
  For i:=1 to 3 do
     For j:=1 to 2 do
        Matriz[i,j]:=random(50)-25;{Заполнение матрицы}
WriteLn('Матрица');
For i:=1 to 3 do begin {Вывод матрицы на экран}
     For j:=1 to 2 do
      Write(Matriz[i,j]:5);
     WriteLn
                end
End.
```

Задание 3. Найти максимальное число матрицы размерности 8x5.

```
Program Matriz_2;
  Var
        M : array [1..8,1..5]  of  integer;
        i,j : integer;
       X : real;
Begin
Randomize;{Изменить    начальное    число    генератора
случайных чисел}
WriteLn('Введите элементы матрицы: ');
For i:=1 to 8 do
     For j:=1 to 5 do
         M[i,j]:=random(50)-25;   {Заполнение матрицы}

WriteLn('Матрица');
For i:=1 to 8 do begin
      For j:=1 to 5 do Write(Matriz[i,j]:5);
      WriteLn
                  end;
```

```
X:=M[1,1];  {Определение начального значения X}
For i:=1 to 8 do
     For j:=1 to 5 do
          If  X<M[i,j] then  X:=M[i,j];
WriteLn('Максимальное число матрицы  ', X:5) ;
ReadLn
End.
```

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вычислить и запомнить сумму и число положительных элементов каждого столбца матрицы. Результаты вывести в виде двух строк.
2. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вычислить и запомнить суммы и число положительных элементов каждой строки матрицы. Результаты вывести в виде двух столбцов.
3. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вычислить сумму и число положительных элементов матрицы, находящихся на главной диагонали и над ней.
4. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вычислить сумму и число положительных элементов матрицы, находящихся под главной диагональю и на ней.
5. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Записать на место отрицательных элементов матрицы нули и вывести ее на экран в общепринятом виде.
6. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Записать на место отрицательных элементов нули, а на место положительных – единицы. Вывести на экран нижнюю треугольную матрицу в общепринятом виде.

7. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вывести на экран нижнюю треугольную матрицу в общепринятом виде. Вычислить суммы столбцов полученной матрицы и записать в строку под каждым столбцом свою сумму.
8. Дана матрица A[10,10]. Заполнить ее случайными числами в диапазоне [-100 . . 100]. Вывести на экран верхнюю треугольную матрицу в общепринятом виде. Вычислить суммы столбцов полученной матрицы и записать в строку под каждым столбцом свою сумму.
9. Определить массив кубов первых ста натуральных чисел и распечатать его в виде матрицы 10x10.
10. Определить массив первых 196 натуральных нечетных чисел, не кратных трем, и распечатать его в виде матрицы 14x14.
11. Определить массив первых 120 натуральных чисел, сумма которых кратна 10, и распечатать его в виде матрицы 10x12.
12. Определить и вывести массивы чисел X и Y, где X=0, 0.2, 0.4, ...20, Y= X^2-20cos(X). Затем вывести в 10 колонок с заголовками сначала положительные элементы массива Y, а затем отрицательные. После таблицы вывести значения Y_min и Y_max.
13. Определить массив Y= sin(X^2)-cos(X), где X= X=0, 0.2, 0.4, ...60. Распечатать в 10 колонок с заголовками сначала номера отрицательных элементов массива, а затем положительных. После таблицы вывести значения Y_min и Y_max.
14. Слить два массива A и B по 100 элементов в массив C из 200 элементов так, чтобы элементы массива A имели в C нечетные номера.
15. Слить два массива A и B по 100 элементов в массив C из 200 элементов так, чтобы элементы массивов A и B чередовались по 10 штук.
16. Слить два массива A и B по 100 элементов в массив C из 200 элементов так, чтобы вначале шли элементы меньше среднего значения по всему массиву C.

Л А Б О Р А Т О Р Н А Я Р А Б О Т А № 9
П Р О Ц Е Д У Р Ы И Ф У Н К Ц И И

Программирование с использованием подпрограмм пользователя

В практике программирования часто встречается ситуация, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменения в нескольких других частях программы.

Для эффективного программирования подобных повторений было введено понятие *подпрограммы.* В языке Паскаль подпрограммы реализуются в виде *процедур и функций.*

Процедура – поименованная часть программы, предназначенная для выполнения определенных действий. Она состоит из заголовка и тела.

По структуре ее можно рассматривать как программу в миниатюре. После однократного описания процедуру можно вызывать по имени из последующего частей программы. Традиционно процедура помещается в главной программе после раздела **var** и перед **begin** программы, хотя такие жёсткие ограничения в современных трансляторах языка Паскаль отсутствуют.

Когда процедура выполнит свою задачу, программа продолжится с оператора, следующего непосредственно за оператором вызова процедуры.

Использование имени процедуры в программе называется *оператором* процедуры или *вызовом* процедуры.

Имя процедуры *не может* находиться в выражении в качестве оператора.

Описание процедуры включает заголовок (имя) и тело процедуры.

Заголовок состоит из зарезервированного слова ***Procedure,*** идентификатора (имени) процедуры и необязательного заключенного в круглые скобки списка формальных параметров с указанием типа каждого параметра. **Формальные параметры** - это наименования переменных, через

которые передается информация из программы в процедуру либо из процедуры в программу.

Формат: ***Procedure*** **<**имя**> {(**формальные параметры**)};**

Пример:

Procedure Korrect ; { не требует формальных параметров }

Procedure Sort(A: byte); { A – формальный параметр }

Имя процедуры – идентификатор, уникальный в пределах программы.

Тело процедуры представляет собой локальный блок, по структуре аналогичный программе: разделы **label**, **const**, **type**, **var** и выполняемую часть (от **begin** до **end**).

Procedure **<**имя**>*;***

<разделы описаний>

begin

<раздел операторов>

end;

Для обращения к процедуре используется оператор вызова процедуры.

Он состоит из идентификатора (имени) процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенных в круглые скобки.

Список параметров может отсутствовать, если процедуре не передается никаких значений.

Sort(a1,b1); { параметры – значения переменных ***}***

Kvadr (14,25,True); { параметры – непосредственные значения ***}***

Sum; { Фактические параметры не указаны, так как в вызываемой процедуре нет параметров***}***

Параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными начальными данными.

Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры

устанавливается взаимно-однозначное соответствие в результате их перебора слева направо.

Количество и тип формальных параметров равны количеству и типу фактических параметров. Если процедура возвращает в программу какие-то значения, соответствующие переменные записываются с использованием перед ними слова **Var.**

Рассмотрим нахождение корней квадратного уравнения $y^2+2y-3=0$.

```
Program S;
Var Y1,Y2: Real;
 Procedure SQ (A,B,C : Real; VAR X1,X2 : Real);
Var D: Real;
begin
    D:=B*B-4*A*C;
    X1:= (-B+Sqrt(D))/(2*A);
    X2:= (-B-Sqrt(D))/(2*A);
end;
Begin {Program}
  SQ(1,2,-3,Y1,Y2);
  WriteLn('Y1=',Y1,'Y2=',Y2);
End.
```

Функция аналогична процедуре, но имеет два отличия:

- функция передает в точку вызова скалярное значение (результат своей работы);
- имя функции *может* входить в выражение как операнд.

Функция, если она встречается в выражении, называется *указателем* функции или *обращением* к функции.

Функция, определенная пользователем, состоит из заголовка и тела функции.

Заголовок содержит зарезервированное слово ***Function,*** идентификатор (имя) функции, заключенный в круглые скобки,

необязательный список формальных параметров и тип возвращаемого функцией значения.

Формат:

Function <имя> ***{(*** Формальные параметры***)} : <***тип результата***>;***

Пример:

Function PRIM (x,y,t : integer) : real ;

Function Logic : boolean;

Имя функции – уникальный в пределах блока идентификатор.

Возвращаемый результат может иметь любой скалярный тип или тип ***string.***

Тело функции представляет собой локальный блок, по структуре аналогичный программе:

Function <имя> ***{(*** Формальные параметры ***)} :*** < тип результата >;

< разделы описаний >

begin

<Раздел операторов>

end;

В разделе операторов должен находиться по крайней мере один оператор, присваивающий идентификатору функции значение.

Если таких присваиваний несколько, то результатом выполнения функции будет значение последнего оператора присваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов.

Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

Формат: <Идентификатор функции> ***{(***Фактические параметры***)};***

Пример:

Write (MinZn(m)); ***{*** Параметр - значение переменной ***}***

C1(9, 67) ; ***{*** Параметры - непосредственно значения ***}***

Summa; *{*Фактические параметры не указаны, так как в вызываемой функции нет формальных параметров*}*

Функции могут возвращать значения целочисленных, вещественных, булевских, литерных, строковых, перечисляемых пользовательских типов.

Пример. Вычисление большего из двух данных чисел можно оформить таким образом.

```
Program S;
Var A,B,C: Real;
Function MAX(X,Y:Real): Real;
begin
      IF X>Y then MAX:=X else MAX:=Y;
end;
Begin {Program}
  WriteLn('Введите два числа');
  ReadLn(A,B);
  C:=MAX(A,B);
  WriteLn('C=',C);
End.
```

ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ

Составить две программы – с использованием функции и с использованием процедуры.

1. Написать программу нахождения среднего арифметического двух чисел, вводимых с клавиатуры.
2. Написать программу нахождения функции трех чисел x_1, x_2, x_3, вводимых с клавиатуры.

$$z = \frac{x_1 + x_2 + x_3}{3}$$

3. Написать программу нахождения произведения двух чисел, вводимых с клавиатуры.

4. Написать программу, определяющую вхождение точки с координатами X и Y внутрь круга с радиусом R.
5. Написать программу нахождения разности двух чисел, вводимых с клавиатуры.
6. Написать программу нахождения суммы целых чисел от 1 до N, где N задается пользователем с клавиатуры.
7. Написать программу, которая вычисляет объем цилиндра. Параметры – радиус и высота цилиндра.
8. Написать программу, которая возвращает максимальное из двух целых чисел, полученных в качестве аргумента.
9. Дана матрица A[10,10]. Вычислить и запомнить сумму и число положительных элементов каждого столбца матрицы.
10. Дана матрица A[10,10]. Вычислить и запомнить суммы и число отрицательных элементов каждой строки матрицы.
11. Дана матрица A[10,10]. Найти минимум каждой строки.
12. Дана матрица A[10,10]. Найти максимум каждой строки.
13. Дана матрица A[10,10]. Найти минимум каждого столбца.
14. Дана матрица A[10,10]. Найти максимум каждого столбца.
15. Дана матрица A[10,10]. Найти число и сумму отрицательных элементов каждого столбца.
16. Дана матрица A[10,10]. Вычислить и запомнить суммы и число положительных элементов каждой строки матрицы.

ЛИТЕРАТУРА

1. Культин Н.Б. Turbo Pascal в задачах и примерах. СПб.: БХВ-Петербург,2002.
2. Бородич Ю.С., Вальвачев А.Н., Кузьмич А.И. Паскаль для персональных компьютеров: Справочное пособие. Мн.: Выш.школа, 1991.
3. Алексеев В.Е., Ваулин А.С., Петрова Г.Б. Вычислительная техника и программирование: Практикум по программированию: Практ. Пособие. М: Высш.шк., 1991.
4. http://cube.h1.ru/pascal/pascal1.html
5. http://rusedu.ru/PrintArticle546.html
6. http://sysopil.narod.ru/prog2.htm
7. http://phmsh.rsc.pp.ru/

Шпехт Ирина Александровна,

доцент кафедры информационных и управляющих систем АмГУ,

канд. техн. наук;

Саакян Рустам Рафикович,

профессор кафедры математического анализа и моделирования АмГУ,

доктор техн. наук.

Информатика (алгоритмизация и основы программирования): Учебно-методическое пособие.