Министерство образования и науки Российской Федерации
ФГАОУ ВПО «Российский государственный
профессионально-педагогический университет»
Учреждение Российской академии образования «Уральское отделение»

С. Б. Петров, С. Н. Ширева

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебное пособие

Допущено Учебно-методическим объединением по профессионально-педагогическому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности 050501.06 – Профессиональное обучение (информатика, вычислительная техника и компьютерные технологии)

Екатеринбург
РГППУ
2011

УДК 004.438(075.8)
ББК В185я73-1
П30

Петров С. Б.

П30 Основы алгоритмизации и программирования: учебное пособие / С. Б. Петров, С. Н. Ширева. Екатеринбург: Изд-во Рос. гос. проф.-пед. ун-та, 2011. 161 с.

ISBN 978-5-8050-0448-4

Рассматриваются примеры решения задач на языке Pascal. Акцент ставится на алгоритм. Для каждой из задач приведена блок-схема решения, что позволяет использовать пособие и для работы на других языках программирования.

Предназначено студентам высших учебных заведений, обучающимся по специальности 050501.06 Профессиональное обучение (информатика, вычислительная техника и компьютерные технологии). Также будет полезно студентам высших и средних специальных учебных заведений других специальностей и старшим школьникам.

УДК 004.438(075.8)
ББК В185я73-1

Рецензенты: доктор физико-математических наук, профессор В. Н. Сыромятников (НВПОУ «Уральский институт экономики, управления и права»), кандидат технических наук, доцент В. В. Вьюхин (ФГАОУ ВПО «Российский государственный профессионально-педагогический университет»)

ISBN 978-5-8050-0448-4

Введение

Данное учебное пособие предназначено для студентов – специалистов в области профессиональной педагогики, специализирующихся в сфере вычислительной техники и компьютерных технологий. Излагаются основы алгоритмизации, иллюстрированные примерами на языке Pascal. При этом учитываются особенности компетенций специалиста указанного профиля в области программирования. И при разработке, и при эксплуатации сложных компьютерных систем ему требуется адаптировать соответствующее программное обеспечение. Он должен уметь написать или изменить какую-то программу, создать новый программный интерфейс для работы с профессионально-ориентированной компьютерной системой, перепрограммировать систему контроля данных, написать новую программу загрузки базы данных и т. д., но совсем не обязан быть профессиональным системным программистом или «хакером». Поэтому основная цель пособия – сформировать у студентов базовые навыки программирования.

Это определило выбор и языка программирования, и предлагаемых студентам тем. Как показывает многолетний опыт, приобретенный авторами в процессе преподавания вводного курса программирования, Pascal в наибольшей степени подходит для достижения учебных целей. Это универсальный язык, для которого характерны экономичность выражения, современные поток управления и структура данных, богатый набор операторов. По разнообразию и количеству средств, предоставляемых программистам, его можно считать одним из самых эффективных языков. Кроме того, Pascal имеет довольно строгий синтаксис, ориентированный на формирование у обучающихся навыков «хорошего стиля программирования», позволяющего существенно сократить количество ошибок, допускаемых при составлении программы. Так, синтаксические ошибки выявляются в Pascal на этапе компиляции. Однако в программе остаются логические ошибки, связанные с ее несоответствием алгоритму решения поставленной задачи. Для обнаружения логических ошибок формального аппарата не существует, вот почему так важно научить студентов способам их обнаружения. Основной акцент в данном пособии делается на методе тестов. Созданная программа выполняется для специально разработанных вариантов исходных данных с известными ответами, называемых тестами. Результаты, полученные на ЭВМ, сравниваются с правильными ответами. Их несовпадение свидетельствует о наличии в программе логической ошибки.

Ориентация на формирование у студентов базовых навыков программирования, а также педагогический опыт авторов, показывающий, что студент, не будучи асом алгоритмизации, не сможет по достоинству оценить возможности объектно-ориентированного программирования, предоставляемые языком Pascal, позволили нам ограничиться небольшим кругом тем. Они способствуют формированию основных компетенций профессионального программиста и позволяют заложить хорошую основу для последующего совершенствования студентов при изучении других компьютерных дисциплин, в рамках которых они будут знакомиться с объектно-ориентированными технологиями. Предлагаемые нами темы охватывают основные виды алгоритмических процессов (линейный, с ветвлением и циклический), а также структуры данных, широко применяемые программистами (переменные, массивы, файлы).

Заключительным этапом изучения каждой учебной темы является выполнение лабораторной работы, по результатам которой оформляется отчет и проходит его защита.

Лабораторная работа должна быть выполнена в соответствии с вариантом, выданным преподавателем. Варианты, содержащиеся в данном пособии, составлены с учетом межпредметных связей и позволяют студентам закрепить знания, полученные при изучении таких дисциплин, как математика, физика, химия, физиология и т. д.

Отчет должен содержать описание решаемой задачи для конкретного варианта, описание алгоритма решения задачи, программу на языке Pascal, тесты, результаты выполнения программы и выводы. Если при решении задачи были допущены ошибки, то необходимо сделать их анализ. Для сложных задач в отчете следует привести план отладки, тесты, результаты отладки по тестам и результат счета основного варианта. Материалы отчета скрепляются, а его страницы нумеруются, начиная с титульного листа.

Защищая отчет, студент должен продемонстрировать работу отлаженной программы в соответствии с вариантом задания и ответить на вопросы преподавателя.

Мы не претендуем на оригинальность изложения теоретического материала по программированию. При создании учебного пособия нами использовались переработанные соответствующим образом материалы из книг, пособий и справочников известных авторов: Н. Вирта, Д. Кнута и др. [2, 4].

Тема 1. ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС

Этапы разработки программы

Разработка любой программы может быть разбита на ряд этапов.

1. Определение входных и выходных данных, требований к программе.

2. Разработка алгоритма.

На этом этапе производится определение последовательности действий, ведущих к решению задачи и запись их в виде блок-схемы.

3. Кодирование (программирование).

Данный этап состоит в переводе алгоритма на язык программирования и в создании *исходного текста программы*. Программа на любом языке состоит из *операторов* – так называются отдельные действия, разрешенные в языке.

4. Компиляция и отладка.

Исходный текст не будет непосредственно исполняться компьютером – для работы программы ее требуется *откомпилировать*, т. е. перевести в машинный код. Программа, которую удалось откомпилировать, не обязательно работает правильно. Она может содержать ошибки, для выявления которых предназначен этап *отладки* – поиска ошибок в программе.

Возможны программные ошибки трех видов:

- *синтаксические* (ошибки в правилах языка);
- *алгоритмические* (ошибки в логике программы);
- *ошибки времени исполнения*, возникающие в процессе работы запущенной программы.

Компилятор способен найти только синтаксические ошибки, для выявления же алгоритмических ошибок служит этап *тестирования* программы. Ошибки времени исполнения возникают как результат некорректных действий пользователя, недопустимых операций над данными (например, попытки извлечь квадратный корень из отрицательного числа, поделить на ноль) или ошибок программного и аппаратного обеспечения ЭВМ.

5. Тестирование.

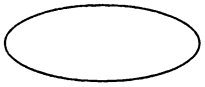
Тестированием называют проверку правильности работы программы на наборах «пробных» (*тестовых*) данных с заранее известным результатом. Конечно же, тестирование всей программы сразу возможно лишь для несложных учебных задач. Реальные программы, как правило, тестируются по частям – отдельными функциями и модулями.

Блок-схема

Блок-схема – это последовательность блоков, предписывающих выполнение определенных операций, и связей между этими блоками. Внутри блоков указывается информация об операциях, подлежащих выполнению.

Блоки соединяются линиями потоков. При соединении блоков следует использовать только вертикальные и горизонтальные линии потоков. Горизонтальные потоки, имеющие направление справа налево, и вертикальные потоки, имеющие направление снизу вверх, должны быть обязательно помечены стрелками.

Любой алгоритм начинается с блока «Начало», а заканчивается блоком «Конец», которые изображаются в виде овала:



Каждому блоку можно поставить в соответствие оператор языка. Исключение составляют только блоки «Начало» и «Конец».

Структура программы

Программа на языке Pascal состоит из заголовка программы и тела программы, за которым следует точка – признак конца программы. В свою очередь, тело программы содержит раздел описаний и раздел операторов:

Program *имя программы*
раздел описаний
Begin
раздел операторов
End.

Раздел операторов имеется в любой программе и является основным. Предшествующие разделы носят характер описаний и не обязательно присутствуют в каждой программе в полном составе.

В простейшей форме программы в разделе описаний имеется раздел описания переменных, который начинается со служебного слова *«var»* и содержит описания всех переменных:

var
переменная1: тип
переменная2: тип
…

Типы данных

Тип – это характеристика данных, задающая границы изменения данных и определяющая множество операций над ними.

Типы данных делятся на простые и структурные. К простым типам данных относятся: целый тип, вещественный тип, логический тип, символьный тип, а также другие не рассматриваемые здесь типы (перечислимый и диапазон).

Целый тип присваивается данным, которые во время работы программы могут принимать лишь целочисленные значения. В современных реализациях языка имеется множество целых типов. В стандарте языка Pascal определен целый тип Integer.

Символьный тип (Char) представляет собой символы раскладки ASCII.

Логический тип (Boolean) может принимать одно из двух значений: True или False.

Данные вещественного типа – это вещественные значения, записанные в памяти в виде чисел с плавающей точкой. Вещественный тип в стандарте языка Pascal – Real. В современных реализациях языка имеются дополнительные вещественные типы.

Выражения и операции

Выражение – это синтаксическая единица, задающая порядок и способ вычисления некоторого значения. Выражение представляет собой последовательность операндов, соединяющихся друг с другом знаками операций. В качестве операндов в конструкции выступают переменные, константы и функции. Операции подразделяются на арифметические, логические и операции отношения.

Операторы языка можно разделить на простые и сложные. Простые операторы не содержат внутри себя других операторов. К простым операторам относятся оператор присваивания, пустой оператор, операторы ввода и вывода.

Оператор присваивания

Оператор присваивания используется для сохранения результата вычисления арифметического выражения в переменной. Он имеет следующий общий вид: *переменная := выражение;*

Знак «:=» читается как «присвоить». Оператор присваивания работает следующим образом: сначала вычисляется выражение, стоящее *справа* от знака «:=» , затем результат записывается в переменную, стоящую *слева* от знака. Например, после выполнения оператора *k* := *k* + *2*; текущее значение переменной *k* увеличится на 2.

Тип переменной слева от знака присваивания должен быть *не младше* типа выражения. В частности, это означает, что если выражение дает целое число, результат можно писать и в целую, и в вещественную переменную; если результат вычисления выражения вещественный, писать его в целую переменную нельзя, так как может произойти потеря точности.

На блок-схеме оператор присваивания соответствует блоку процесса:

Оператор ввода

Базовая форма ***оператора ввода*** позволяет пользователю ввести с клавиатуры значения одной или нескольких переменных. Оператор ввода с клавиатуры может быть записан в одной из следующих форм: *read(список_переменных)* или *readln(список_переменных).*

Имена переменных в списке перечисляются через запятую. Здесь и далее список данных, передаваемых любому оператору (а позднее и написанным нами подпрограммам), мы будем называть *параметрами.* Таким образом, параметрами оператора (точнее, *стандартной процедуры*) read являются имена переменных, описанных ранее в разделе var.

По достижении оператора ввода выполнение программы останавливается и ожидается ввод данных пользователем. Вводимые значения переменных разделяются пробелом или переводом строки (нажатием клавиши Enter). После ввода значений всех переменных из списка работа программы продолжается со следующего оператора.

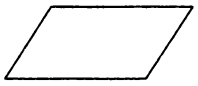
Оператор *readln* отличается от *read* только тем, что все переменные должны быть введены в одну строку экрана, клавиша Enter нажимается один раз по окончании ввода. Форма записи *readln* используется, в основном, для ввода строк текста, для ввода числовых значений лучше использовать *read*, так как в этом случае пользователь может вводить данные более свободно (и в одну, и в несколько строк экрана).

Если пользователь вводит данные недопустимого типа (например, строку текста вместо числа), то выводится системное сообщение об ошибке и работа программы прерывается.

Как правило, перед оператором ввода ставится оператор вывода, служащий приглашением к вводу и поясняющий пользователю, что именно следует сделать.

Если программа выполняется в консольном приложении, для задержки экрана с выводом данных также используется оператор *readln*.

На блок-схеме оператор ввода изображается в виде параллелограмма:



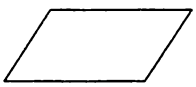
Оператор вывода

Базовая форма ***оператора вывода*** позволяет отобразить на экране значения переменных или констант, а также строки текста в апострофах. Оператор записывается в одной из следующих форм: *write(список)* или *writeln(список)*.

Элементы списка перечисляются через запятую и выводятся в пользовательское консольное окно программы. Вещественные значения выводятся в экспоненциальной форме. Строки выводятся «как есть». После вывода работа программы продолжается со следующего оператора.

Оператор *writeln* отличается от *write* тем, что после вывода значения последнего элемента списка выполняется перевод курсора на следующую строку экрана.

На блок-схеме оператор вывода также изображается в виде параллелограмма:



Управление выводом данных

В операторе *write* или *writeln* вещественное значение (а также целое или строковое) зачастую удобнее записывать в следующем виде: *переменная:ширина:точность*

Здесь *ширина* – целое положительное число, определяющее, сколько экранных позиций отводится для вывода всего числа. Ширина определена для числовых значений любого типа и строк.

Точность – целое положительное число, определяющее, сколько цифр из ширины отводится на вывод дробной части числа. Значение точности определено только для вещественных чисел. Оно не учитывает позицию десятичной точки. Разумные значения точности – от *0* до *ширина – 2* включительно. Недопустимые значения ширины и точности не будут учтены при выводе.

Лабораторная работа 1. Арифметические выражения

Теория

Линейным принято называть вычислительный процесс, в котором операции выполняются последовательно, в порядке их записи. Каждая операция является самостоятельной, независимой от каких-либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности. Основу программы линейного вычислительного процесса составляют операторы присваивания, ввода и вывода данных.

Линейные вычислительные процессы имеют место, например, при вычислении арифметических выражений, когда есть конкретные числовые данные и над ними выполняются соответствующие условию задачи действия.

В арифметических выражениях операции могут применяться только к операндам целых и вещественных типов: «+» – сложение; «–» – вычитание; «*» – умножение; «/» – вещественное деление; *div* – целочисленное деление; *mod* – остаток от деления целых чисел.

В результате вещественного деления даже при делении целых чисел получается вещественное значение. Операции *div* и *mod* выполняются только над целыми числами.

При вычислении учитывается приоритет операций: в первую очередь выполняются операции деления и умножения в том порядке, в каком они написаны, затем сложение и вычитание. Для изменения порядка вычисления используются круглые скобки.

В качестве операндов могут выступать стандартные арифметические функции (табл. 1).

Таблица 1

Функции языка Pascal

Функция	Значение функции
Pi	Число π
Abs(*x*)	Модуль *x*
Arctan(*x*)	Арктангенс *x* (радианы)
Cos(*x*)	Косинус *x* (*x* в радианах)
Sin(*x*)	Синус *x* (*x* в радианах)
Exp(*x*)	*ex* – экспонента
Ln(*x*)	Натуральный логарифм *x*
Sqr(*x*)	Квадрат *x*
Sqrt(*x*)	Квадратный корень *x*
Int(*x*)	Целая часть *x*
Frac(*x*)	Дробная часть *x*
Round(*x*)	Округление до ближайшего целого
Trunc(*x*)	Ближайшее целое, не превышающее *x* по модулю
Random	Псевдослучайное число в интервале [0, 1)
Random(*x*)	Псевдослучайное число в интервале [0, *x*)

Операция возведения в степень в языке Pascal отсутствует. При возведении в степень следует использовать специальную формулу.

Формулы, с помощью которых можно вычислять функции, отсутствующие в языке Pascal, приведены в табл. 2.

Таблица 2

Формулы для вычисления функций, отсутствующих в языке Pascal

Функция	Формула для вычисления
x^y	*exp*(*y***ln*(*x*))
tg(*x*)	*sin*(*x*)/*cos*(*x*)
$log_a b$	*ln*(*b*)/*ln*(*a*)
arcsin(*x*)	*arctan*(*x*/*sqrt*(1 – *x***x*))
arccos(*x*)	P*i*/2 – *arctan*(*x*/*sqrt*(1 – *x***x*))
arcctg(*x*)	P*i*/2 – *arctan*(*x*)

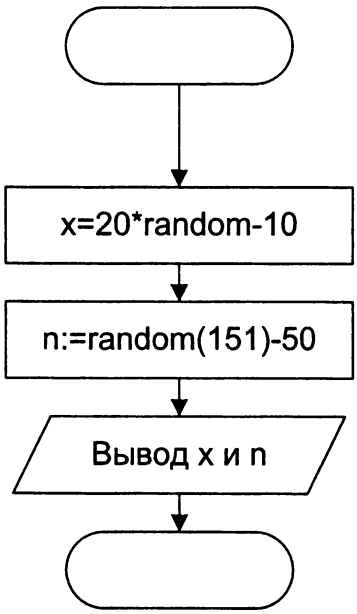
Примеры

Пример 1

Сгенерировать вещественное число в интервале от –10 до 10 и целое число в интервале от –50 до 100.

Исходные данные: интервал для вещественных чисел от –10 до 10, интервал для целых чисел от – 50 до 100.

Результат: x – вещественное число, n – целое число.




```
program rnd;
 var x: real;
     n: integer;
begin

      randomize;
      x:=20*random-10;

      n:=random(151)-50

      writeln('x=',x:5:1, 'n=', n:4);
      readln;

end.
```

Пример 2

Найти z: $z = |a - 10| \cdot \log_2(4 - b) + 2(b - 10) + \sqrt[5]{a^4}$,

где $a = b^{-0,25}\arccos 0,6 - (d\sqrt{d})^{-\frac{b}{3}} tgb$;

$$b = \frac{d(1 - \cos 2\alpha + \sin 2\alpha)}{1 + \cos 2\alpha + \sin 2\alpha} + d;$$

$$d = \frac{\sin\alpha}{1 + \dfrac{\cos k + 1}{\operatorname{tg}^2 15 \cdot k}},$$

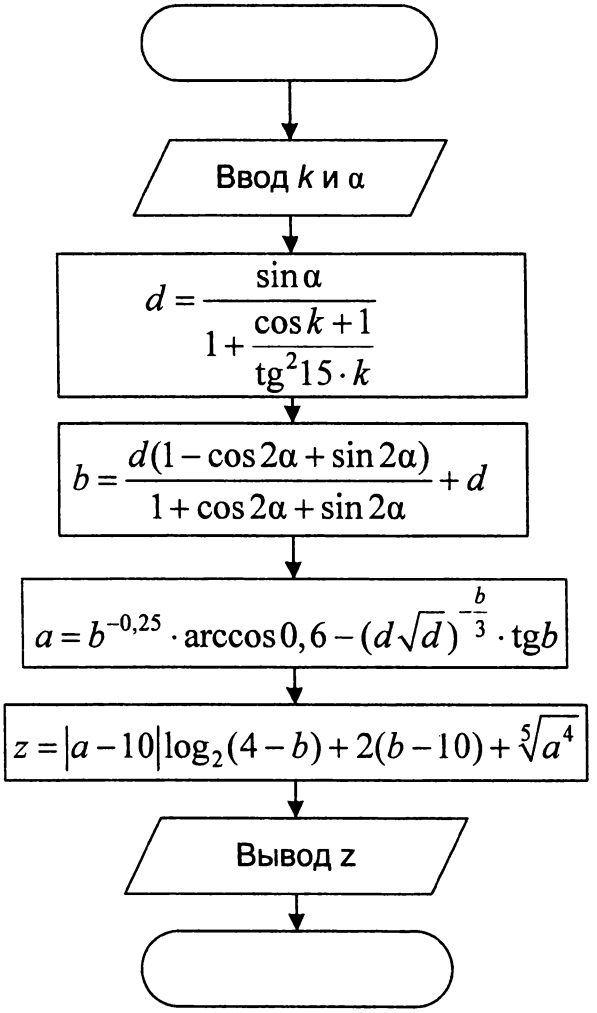
для произвольных k и α.

Исходные данные: k – вещественный тип, α будет обозначаться как *alf* – вещественный тип.

Результат: z – вещественный тип.

Следует учитывать, что при вычислении выражения все переменные, участвующие в данном выражении, должны быть известны. Это диктует последовательность вычислений.

Тестовый пример: при $k = 1$ и $\alpha = 1$ $z = 10{,}81$.




```
program arif;
var k, alf, d,b,a,z: real;
  begin

write('k='); readln(k);
write('alf='); readln(alf);

d:=sin(alf)/(1+(cos(k)+1)/
     (sqr(sin(15)/cos(15))*k));

b:=d*(1-cos(2*alf)+sin(2*alf))/
     (1+cos(2*alf)+sin(2*alf))+d;

a:=exp(-0.25*ln(b))*
     (pi/2-arctan(0.6/sqrt(1-0.6*0.6))-
     exp(-b/3*ln(d*sqrt(d)))*sin(b)/cos(b);

z:=abs(a-10)*ln(4-b)/ln(2)+2*(b-10)+
    exp(1/5*ln(a*a*a*a));

writeln('z=',z:7:2);
readln;

end.
```

Контрольные вопросы

1. Что обозначает каждая цифра после двоеточия в операторе вывода?
2. Какой тип имеет результат выражения 10/2?
3. Какие ограничения существуют при возведении числа в произвольную степень?
4. Выражение $\frac{1}{2a}$ должно быть записано как 1/(2*a). Почему знаменатель заключен в круглые скобки?
5. Почему для ввода данных требуется два оператора: *write* и *readln*?

6. Можно ли в примере 2 изменить порядок вычислений? Например, сначала вычислить a, затем – b?

7. Почему в программе перед оператором *end* стоит оператор *readln*?

Задания для лабораторной работы

1. Найти t: $t = c - ae^{-2} + \operatorname{arctg} k$,

где $a = 3^{-c}\dfrac{\sin\pi \cdot k}{\pi k}$;

$k = \sqrt[3]{\cos^2 x + |\sin^2 c|}$;

$c = \sqrt{\lg\dfrac{1}{|\cos x|} + y}$,

для произвольных x и y.

2. Найти U: $U = x + y + e^{-wvf}$,

где $w = f + v + \ln|x \cdot y \cdot z|$;

$y = \sqrt[3]{|\sin^2(\cos f) + x|}$;

$f = -e^{x\sqrt{|y|}} + z$,

для произвольных x, y и z.

3. Найти m: $m = \sqrt[3]{|n \cdot x \cdot a|}$,

где $n = b\sin x + t$;

$t = \ln b - e^a$;

$b = \sqrt{\operatorname{tg}\dfrac{|x|}{|a|} + x}$,

для произвольных a и x.

4. Найти i: $i = \ln\sqrt[k]{|m + n|}$,

где $k = m^{msinb}$;

$b = \sqrt[3]{\operatorname{tg}^2(m \cdot d)}$;

$n = e^{d-m}$,

для произвольных d и m.

5. Найти w: $w = \sqrt{|a^m \cdot n^b|}$,

где $a = n\sin^4 m$;

$n = \sqrt[m]{b \cdot k}$;

$b = e^k$,

для произвольных m и k.

6. Найти p: $p = g + \frac{f}{s} - e^{\sqrt{|s|}}$,

где $s = f + \sin g^t$;

$g = t^3 + \sqrt[5]{|f \cdot t - 5|}$;

$t = f + \cos^2 k$,

для произвольных f и k.

7. Найти a: $a = e^b e^c e^d$,

где $b = \cos^2 c + \sin^2 d$;

$c = d^{\sqrt{xy}} + e^d$;

$d = \frac{(e^x)^{\frac{1}{y}} - \lg|xy|}{x + y}$,

для произвольных x и y.

8. Найти q: $q = tgp^t$,

где $p = \sqrt[5]{|acb|}$;

$t = \sin^a b^2$;

$c = -e^{\sqrt{a+b+5}}$,

для произвольных a и b.

9. Найти x: $x = \frac{a + b - e^{2c}}{c + \frac{d}{f + a}}$,

где $d = c^{\sin a + \cos b}$;

$f = \frac{-b + \sqrt{|b^2 - 4ac|}}{ac}$;

$c = \sqrt[3]{\text{tg}a^b}$,

для произвольных a и b.

10. Найти z: $z = \text{arctg}y + \ln|x - 1|$,

где $x = \frac{b \cdot \sqrt{a^2 - 1}}{a^2 + b_2} - 2{,}47a$;

$y = \sin^2(\sqrt[3]{x}) - e^{0{,}4(1 + a)}$;

$a = 5\cos c^3$,

для произвольных c и b.

11. Найти v: $v = \sqrt{2\mathrm{tg}u - \frac{1}{u}} + 2\sqrt{m^2 + n^2}$,

где $u = \frac{2(1 - \cos 2a + \sin 2a)}{1 + \cos 2b + \sin 2b}$;

$m = \ln|\mathrm{arctg}a + \mathrm{arctg}b| + a^b$;

$n = \lg\sqrt{2a} + 3\lg\frac{ab}{a + b}$,

для произвольных a и b.

12. Найти f: $f = 4^{3\ln|p + t|} + \sqrt{2ptga}$,

где $p = \frac{\cos^3 x}{\sin^2(1 + \mathrm{tg}^2 a)}$;

$t = e^{\sqrt{x+a}} + a^{2b}$;

$x = \frac{a \cdot \sin a}{b \cdot \cos b}$,

для произвольных a и b.

Тема 2. РАЗВЕТВЛЯЮЩИЙСЯ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС

Разветвляющимся называется алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи – в зависимости от выполнения условий. В отличие от линейных алгоритмов, где команды выполняются последовательно одна за другой, в разветвляющиеся алгоритмы входит условие, в зависимости от выполнения или невыполнения которого реализуется та или иная последовательность действий.

Разветвляющиеся алгоритмы можно осуществить с помощью двух операторов: условного оператора и оператора выбора.

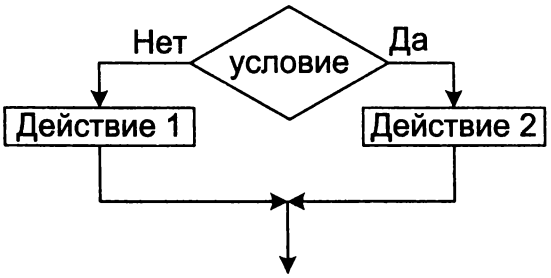
Лабораторная работа 2. Условный оператор

Теория

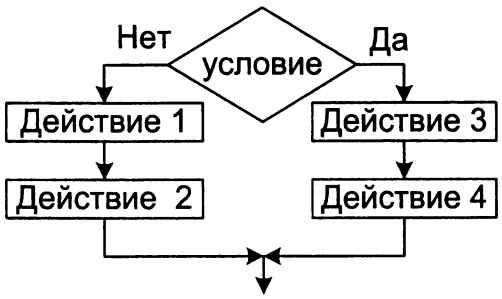
Назначение условного оператора – обеспечить ветвление алгоритма в зависимости от выполнения некоторого условия.

Блок проверки условия (блок принятия решения) изображается в виде ромба с двумя выходами – «Да» и «Нет»:

В языке Pascal условный оператор имеет следующий вид: если условие, записанное после служебного слова *if*, верно, то выполняется действие 1, иначе – действие 2.

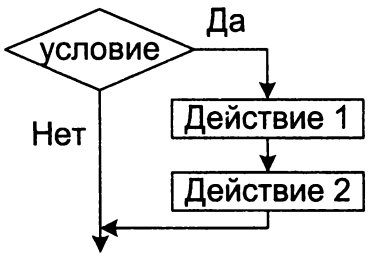


Если на ветках «Да» или «Нет» надо выполнить несколько действий, эти операторы следует заключать в операторные скобки *begin* и *end*:




```
if условие then
    begin
      Действие1;
      Действие2;
    end
  else
    begin
      Действие3;
      Действие4;
    end;
```

Существует неполный вариант оператора *if*, когда ветка «Нет» пустая:




```
if условие
    then begin
           Действие1;
           Действие2;
         end;
```

Условие представляет собой логическое выражение, состоящее из одного или нескольких отношений. Операция отношения выполняет сравнение двух операндов. Результат операции отношения – значение булевского типа *true* или *false*.

Основные операции отношения: «=» – равно; «< >» – не равно; «>» – больше; «<» – меньше; «>=» – больше или равно; «<=» – меньше или равно.

Если условие имеет более сложный вид, т. е. содержит несколько отношений, оно состоит из логических операций, которые и объединяют отношения. В Pascal имеются четыре логические операции: *Not A*, *A and B*, *A or B*, *A xor B* (табл. 3).

Таблица 3

Логические операции

A	*B*	*Not A*	*A and B*	*A or B*	*A xor B*
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

Следует помнить, что в языке Pascal операции отношения имеют самый низкий приоритет. Поэтому каждая операция отношения должна заключаться в круглые скобки. Например: (*a*>6) *and* (*b*<0).

Приоритет логических операций следующий: самый высокий приоритет у операции *not*, затем у операции *and*, а у операций *or* и *xor* самый низкий приоритет среди логических операций.

При использовании оператора *if* важно помнить следующее:

• если на какой-нибудь ветке необходимо выполнить несколько действий, эти действия следует заключать в операторные скобки;

• если на ветке «Да» используются операторные скобки, перед служебным словом *else* ставить точку с запятой нельзя;

• в условии нельзя записывать двухстороннее условие, например, 5<=*x*<=10 (часто его и не требуется записывать, так как часть его уже проверялась на предыдущих ветках). Но если избежать этого невозможно, следует записать двухстороннее условие как сложное: (*x*>=5) *and* (*x*<=10);

• если условный оператор содержит другой условный оператор, то ветка *else* относится к ближайшему условию. Поэтому если при вложении операторов *if* какой-то оператор имеет неполную форму, рекомендуется использовать операторные скобки;

• если выражение, которое фиксируется в условии, слишком длинное, имеет смысл записать условие как отдельный оператор присваивания. В этом случае следует дополнительно объявить переменную булевского типа и записать туда это выражение.

Примеры

Пример 1

Вычислить A: $A = \dfrac{B + C}{2C}$,

где $C = 4B - 20$,

для произвольного B.

Исходные данные: B – вещественного типа.

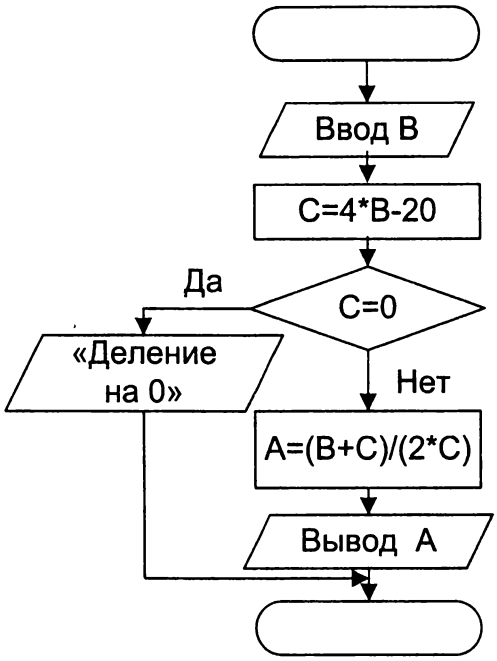
Результат: A – вещественного типа.

Сначала следует вычислить C – часть выражения для вычисления A. При вычислении A может возникнуть ошибка – необходимость деления на ноль. Следовательно, перед началом деления нужно выполнить проверку. Если знаменатель равен нулю – прекратить вычисление и выйти из программы.

Тестовый пример:

a) при $B = 1$ $A = 0{,}468$;

b) при $B = 5$ выводится сообщение «Деление на ноль».




```
program arif;
  var A, B, C: real;
begin
write('B=');
realln(B);

C:= 4*B-20:

 if C=0 then
            begin
               writeln('Деление на ноль');
               exit;
             end;
 A:=(B+C)/(2*C);


 writeln('A=', A:6:2);


 end.
```

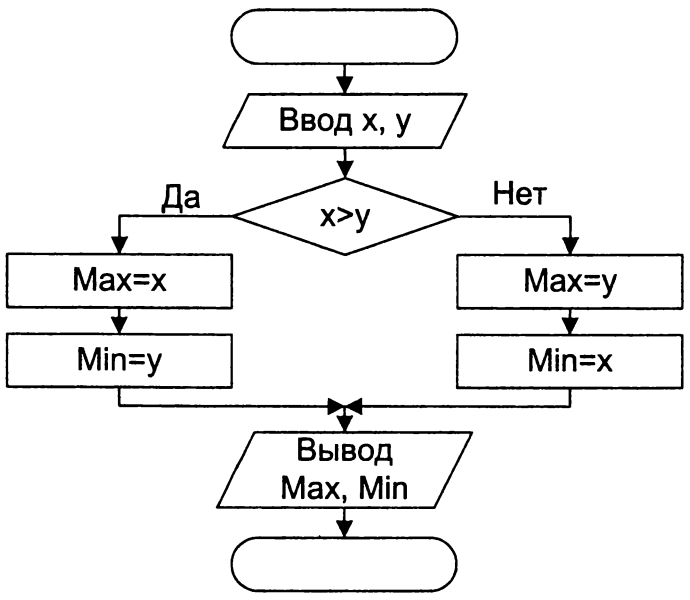
Пример 2

Даны два произвольных числа. Определить максимальное и минимальное из них.

Исходные данные: x и y – вещественного типа.

Результат: Max и Min – вещественного типа.

Тестовый пример: при $x = 5$, $y = 10$ Max = 10, Min = 5.

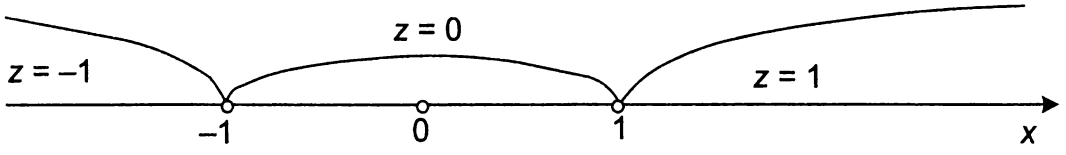



```
program Min_Max;
  var x, y, Max, Min :real;
begin
  write ('Введите x,y');
  readln(x,y);
if x>y then
          begin Max:=x;
                  Min:=y;
          end
      else
           begin Max:=y;
                   Min:=x;
           end;
writeln('Max=', Max:6:2,
            'Min=',Min:6:2);


end.
```

Пример 3

Вычислить z: $z = \begin{cases} -1, \text{если } x < -1, \\ 0, \text{если } -1 \geqslant x \geqslant 1, \\ 1, \text{если } x > 1, \end{cases}$

для произвольного x.



Исходные данные: x – вещественного типа.

Результат: z – вещественного типа.

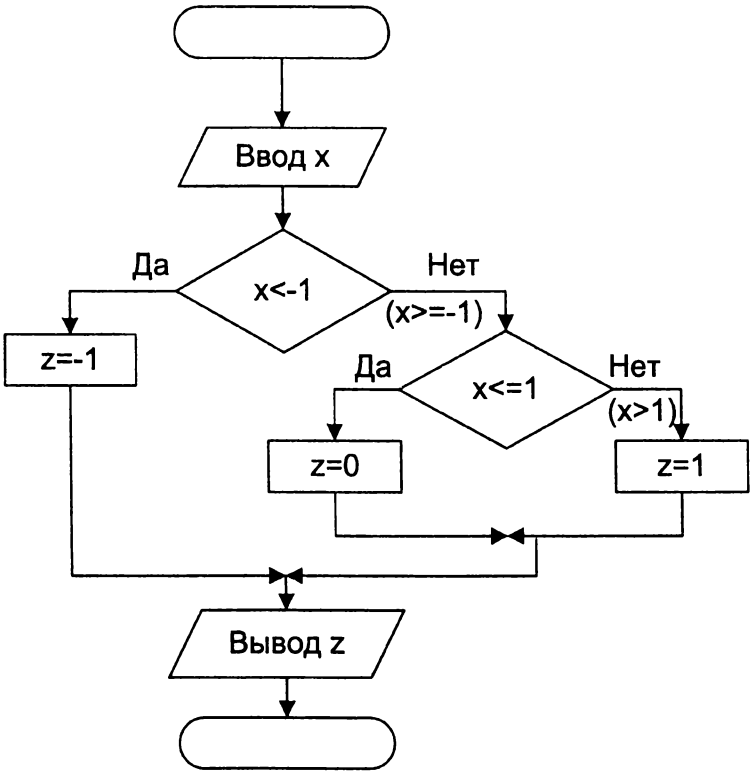
Необходимо определить, в какую часть числовой оси попадает x.

Тестовый пример:

a) при $x = -10$ $z = -1$;

b) при $x = 1$ $z = 0$;

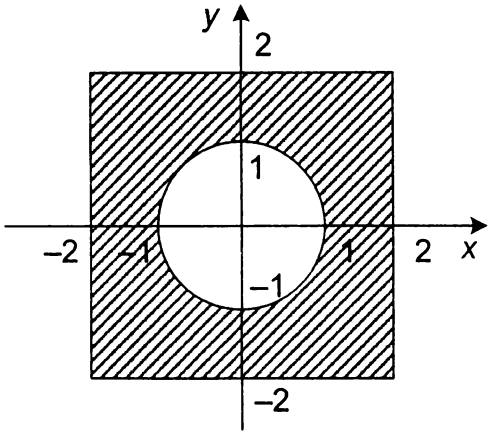
c) при $x = 5$ $z = 1$.




```
program z;
   var x: real;
       z: integer;
begin
   write('x='); readln(x);


if x<-1 then
                z:=-1
           else
              if x<=1 then
                          z:=0
                       else
                          z:=1;


writeln('z= ',z);


end.
```

Пример 4

Определить, попадает ли точка с координатами (a, b) в заштрихованную область, представленную на рисунке.



Исходные данные: a и b – координаты точки – вещественного типа, радиус окружности – 1, половина стороны квадрата – 2.

Результатом будет сообщение, попадает ли точка в данную область.

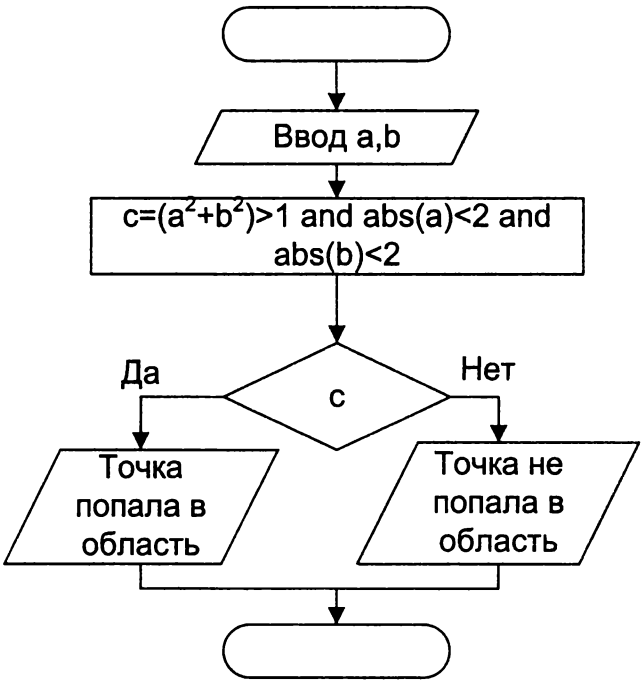
Тестовый пример:

a) при $x = 0{,}5$, $y = 0{,}5$ точка не попала в область;

b) при $x = 1{,}2$, $y = -1{,}5$ точка попала в область;

c) при $x = 3$, $y = -4$ точка не попала в область.

Формула окружности $x^2 + y^2 = r^2$, где r – радиус окружности. Для точек вне окружности выполняется неравенство $x^2 + y^2 > r^2$, если центр окружности имеет координаты (0, 0). В квадрат попадают точки, для которых выполняется условие $|x| < h/2$ и $|y| < h/2$, где h – сторона квадрата и центр квадрата имеет координаты (0, 0). Чтобы точка попала в заштрихованную область, все эти условия должны выполняться одновременно.




```
program usl;
 var a,b: real;
      c: boolean;
begin
   write('Введите a и b'); readln(a,b);
{вычисление условия попадания
в область}
    c:=(a*a+b*b>1) and (abs(a)<2)
and (abs(b)<2);
if c then
 writeln('Точка попала в область')
  else
   writeln('Точка попала в область');
 readln;



 end.
```

Пример 5

Дано целое число. Определить, является ли последняя цифра данного числа квадратом другого числа.

Исходные данные: заданное число x – целого типа.

Результатом будет сообщение, является ли последняя цифра квадратом другого числа.

Среди чисел, меньших 10, имеется только два числа, которые являются квадратами другого числа, – это 4 и 9. Если заданное число больше 9, то надо выделить последнюю цифру. Обозначим эту цифру k. k можно получить как остаток от деления данного числа на 10. Если заданное число меньше 10, то k и есть данное число.

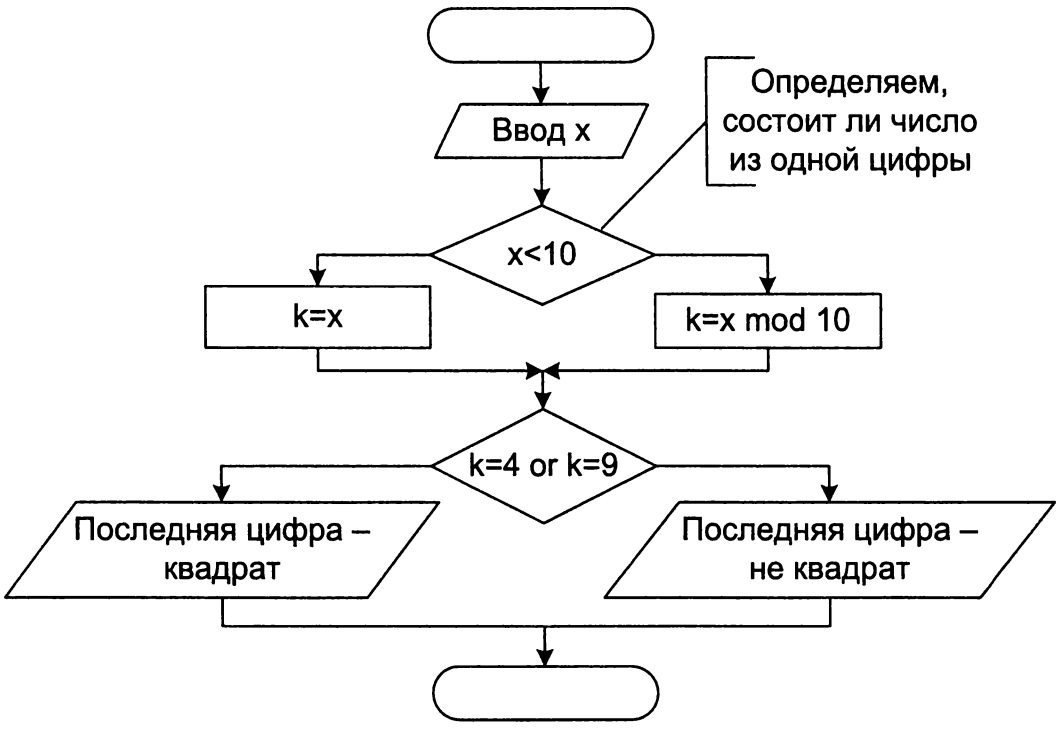
Тестовый пример[1]**:**

a) при $x = 8$ последняя цифра не квадрат;

b) при $x = 9$ последняя цифра – квадрат;

c) при $x = 41$ последняя цифра не квадрат;

d) при $x = 129$ последняя цифра – квадрат.



[1] Здесь и далее для некоторых примеров листинг не приводится – в тех случаях, когда он может быть записан по аналогии с предыдущими задачами.

Контрольные вопросы

1. В какой ситуации в условном операторе используется составной оператор?

2. Почему при записи сложного условия выражения отношения заключаются в круглые скобки?

3. Расставьте в порядке понижения приоритета следующие операции: +, *and*, >=, *or*, *, <, *div*.

4. Почему в примере 3 нет проверки ситуации $x > 1$?

5. Изобразите блок-схему примера 5 без использования сложного условия.

6. По какой ветке будет выполняться алгоритм примера 2, если x и y равны?

Задания для лабораторной работы

1. Рассчет оплаты за электричество происходит следующим образом: если израсходовано меньше A кВт·ч, то оплата составляет X р. за киловатт-час, а каждый киловатт-час сверх нормы стоит Y р. Рассчитать плату за электричество.

2. Определить правильность даты, введенной с клавиатуры (число от 1 до 31, месяц от 1 до 12).

3. По введенному возрасту указать, к какой группе относится человек: дошкольник, ученик, работник, пенсионер.

4. Заданы размеры прямоугольного отверстия – X и Y. Определить, пройдет ли кирпич размерами A, B и C в это отверстие.

5. Вычислить число и месяц дня в невисокосном году по его номеру.

6. Для определения нормального веса вычисляется индекс массы тела (ИМТ) по следующей формуле: вес в килограммах делится на рост в метрах, возведенный в квадрат. Нормальный вес – при ИМТ от 19,5 до 24,9; худоба – при ИМТ ниже 19,5, избыточный вес – при ИМТ от 25 до 27,9, ожирение – при ИМТ больше 28. Определить ИМТ для произвольных значений веса и роста.

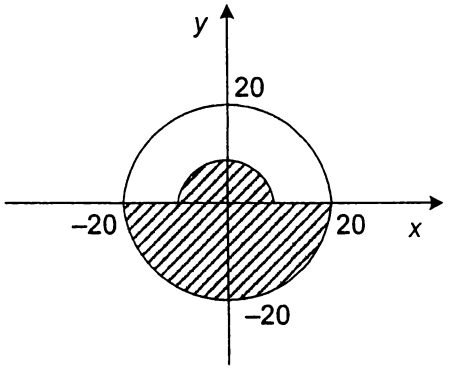
7. Даны a и b. Если $a < 0$ и $b > 0$, то вычислить y по формуле $y = a^2 + 2b$. Иначе

$$y = \begin{cases} 2x^2 - 5x - 6, & \text{если } x > 5, \\ \dfrac{x}{10} - 3, & \text{если } x = 5, \\ 2x - x^2 + 10, & \text{если } x < 5, \end{cases}$$

для произвольного x.

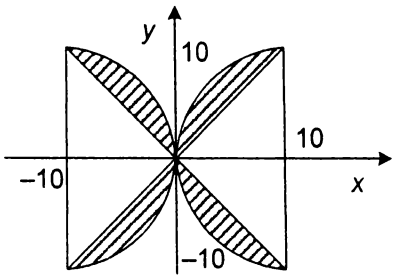
8. Даны две окружности с радиусами R_1 и R_2 с центрами в точках (x_1, y_1) и (x_2, y_2). Определить характер расположения окружностей: пересекаются, касаются, не пересекаются. (Указание: необходимо вычислить расстояние между этими точками и сравнить их с суммой радиусов.)

9. Даны числа a и b. Определить, попадет ли точка с координатами (a, b) в заштрихованную область, и вывести на экран соответствующее сообщение.



Если точка попадет в заштрихованную область, присвоить z максимальное значение из a и b, если точка попадает в незаштрихованную часть круга, присвоить z минимальное значение из a и b.

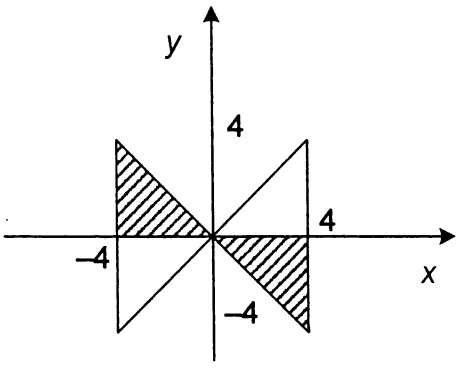
Если точка не попадает в круг, вычислить z по формуле $z = 0{,}5a - b$.



10. Даны числа a и b. Определить, попадет ли точка с координатами (a, b) в заштрихованную область и вывести на экран соответствующее сообщение.

11. Ввести три числа: x, y, z. Если сумма трех попарно различных чисел x, y, z меньше единицы, то меньшее из x и y заменить полусуммой y и z, иначе большее из x и z заменить на y^3.

12. Даны числа a и b. Определить, попадет ли точка с координатами (a, b) внутрь фигуры, и вывести на экран соответствующее сообщение.



Если точка попадает в заштрихованную область, вычислить z согласно условию

$$z = \begin{cases} 4a - 5b, \text{ если } b < 0, \\ 25 - b, \text{ если } b \geqslant 0. \end{cases}$$

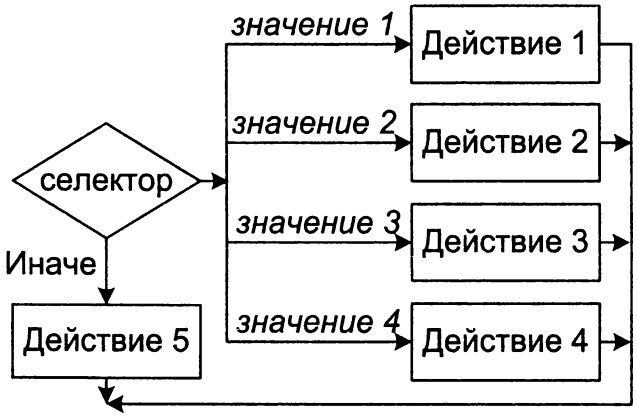
Если точка попадает в незаштрихованную часть фигуры, вычислить z по формуле $z = a - b$.

Если точка не попадает внутрь фигуры, вычислить z по формуле $z = ab$.

Лабораторная работа 3. Оператор выбора

Теория

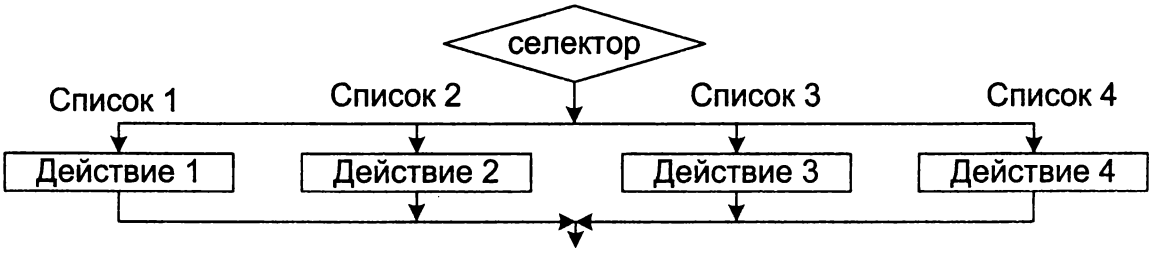
Оператор выбора предназначен для организации выбора одной из любого количества ветвей алгоритма в зависимости от значения некоторого выражения. Оператор выбора обычно используется тогда, когда требуется выполнить множество проверок и использование вложенных операторов *if* слишком громоздко. В этом случае для оператора выбора ГОСТ 19. 701–90 [7] предлагает следующую блок-схему:




```
case селектор of
      значение 1: Действие1;
      значение 2: Действие2;
      значение 3: Действие3;
      значение 4: Действие4
   else           Действие5
end;
```

Точно так же, как и в условном операторе, если на какой-нибудь ветке надо выполнять несколько действий, следует использовать операторы *begin* и *end*.

В операторе выбора на каждой ветке можно проверять не одно значение, а целый список. Список может быть оформлен и как перечень возможных значений (через запятую), и как интервал: нач. знач. … кон. знач.[1] В операторе выбора может отсутствовать ветка *else*. В этом случае, если селектор не соответствует ни одному значению, никаких действий выполняться не будет:



У оператора выбора имеется существенное ограничение: селектор в операторе выбора должен иметь порядковый тип.

[1] Здесь и далее: нач. знач. – начальное значение; кон. знач. – конечное значение.

Примеры

Пример 1

Ввести числовые оценки, вывести текстовые оценки.

Исходные данные: *ot* – числовая оценка – целый тип.

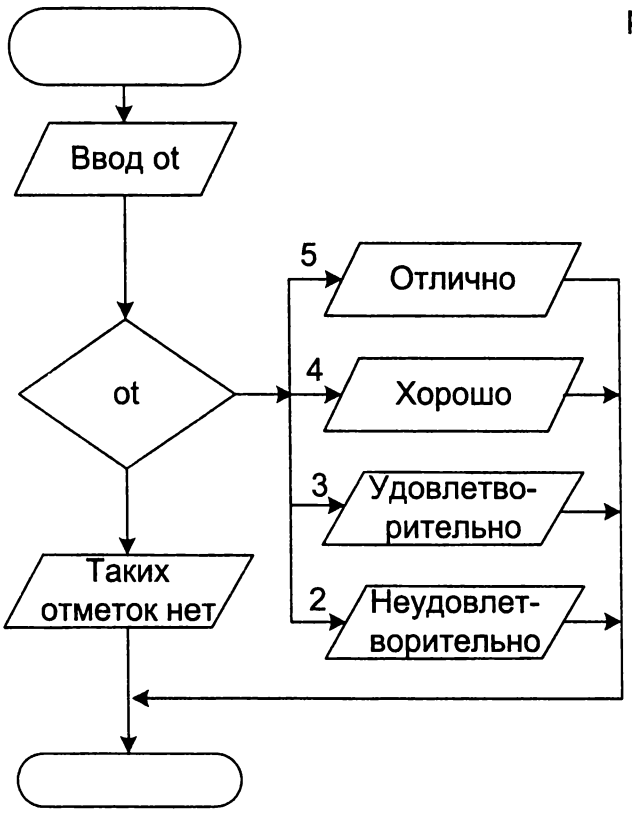
Результатом является вывод оценки в виде текста.

Тестовый пример:

a) при *ot* = 4 вывод «Хорошо»;

b) при *ot* = 4 вывод «Таких отметок нет»;

c) при *ot* = 3 вывод «Удовлетворительно».




```
program otmetka;
 var ot:integer;
  begin
     write('Введите отметку');
     readln(ot);

  case ot of
     5: writeln('Отлично');

     4: writeln('Хорошо');

     3: writeln('Удовлетворительно');

     2: writeln('Неудовлетворительно')
        else
             writeln('Таких отметок нет')
      end;
  end.
```

Пример 2

В восточном календаре (шестидесятилетний календарный цикл) каждый год является не только годом какого-нибудь животного, но и относится к определенной стихии. Каждая стихия охватывает два года подряд. Стихии расположены в следующем порядке: Дерево, Огонь, Земля, Металл, Вода. 2004 год – это первый год стихии Дерева.

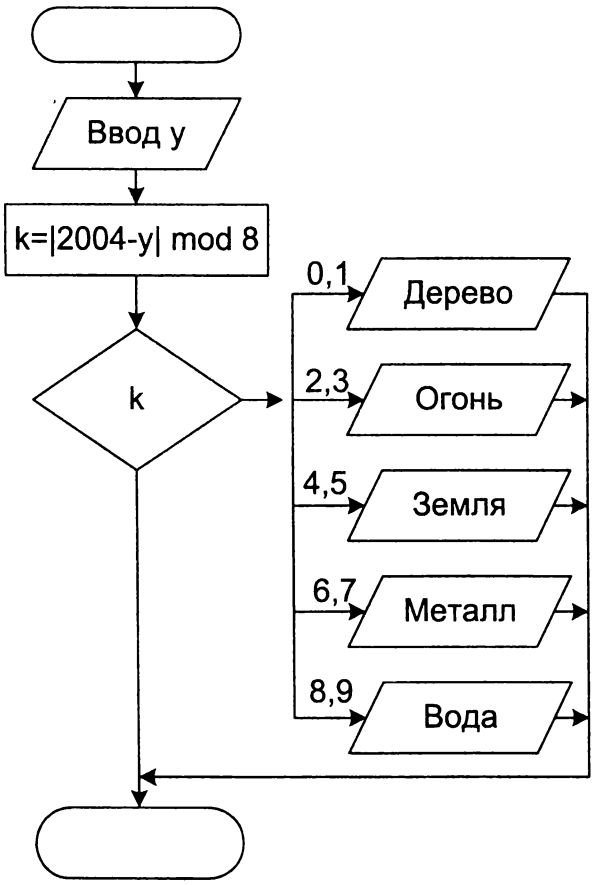
Составить программу, в которой по году определить, к какой стихии он относится. Разработать оптимальный вариант программы.

Исходные данные: y – год – целый тип.

Результатом является вывод названия соответствующей году стихии.

Для определения стихии надо найти разность по модулю между введенным годом и 2004. Затем найти остаток от деления этой разности на 10. Если остаток 0 или 1, то эта стихия – Дерево, если 2 или 3 – Огонь, если 4 или 5 – Земля, если 6 или 7 – Металл, если 8 или 9 – Вода.

Тестовый пример: при $y = 2003$ вывод «Вода».




```
program stihii;
 var y,k: integer;
begin
   write('Введите год'); readln(y);

k:=abs(2004-y) mod 8;
case k of
   0,1: writeln('Дерево');

   2,3: writeln('Огонь');

   4,5: writeln('Земля');

   6,7: writeln('Металл');

   8,9: writeln('Вода');
end;
   readln;

end.
```

Пример 3

Ввести день и месяц текущего года. Проверить правильность введенной даты.

Исходные данные: m – номер месяца – целый тип; d – номер дня – целый тип.

Результатом является сообщение «Правильно» при верно введенной дате или «Неправильно» при неверном введении.

В данной задаче следует учитывать, что 1, 3, 5, 7, 8, 10 и 12-й месяцы имеют 31 день, а 4, 6, 9 и 11-й месяцы – 30 дней. Для 2-го месяца важно, является ли текущий год високосным.

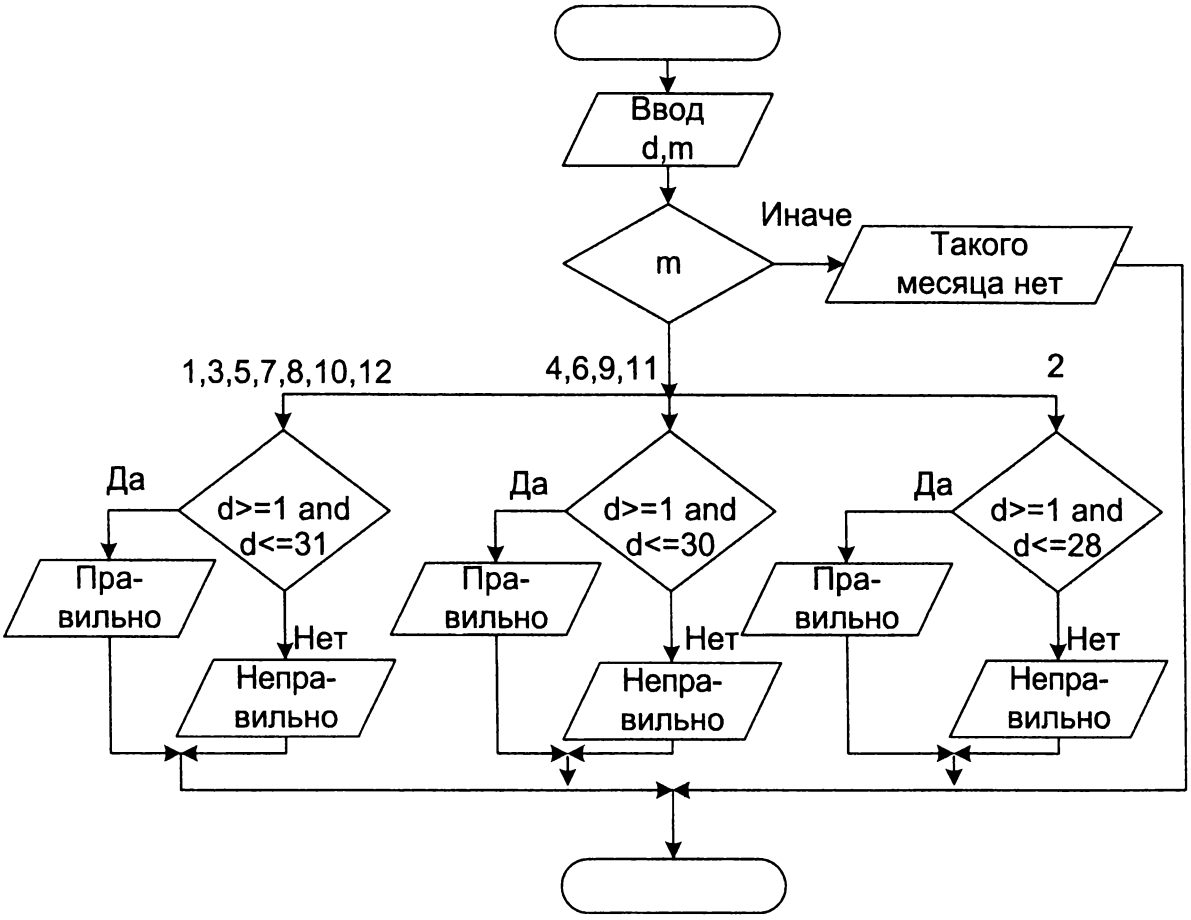
Тестовый пример:

a) при $d = 40$, $m = 5$ вывод «Неправильно»;

b) при $d = 31$, $m = 5$ вывод «Правильно»;

c) при $d = 10$, $m = 5$ вывод «Такого месяца нет»;

d) при $d = 30$, $m = 2$ вывод «Неправильно».



Пример 4

Ввести символ и указать, к какой из групп он относится – группе букв, цифр, знаков арифметических операций или других символов.

Исходные данные: c – введенный символ – символьный тип.

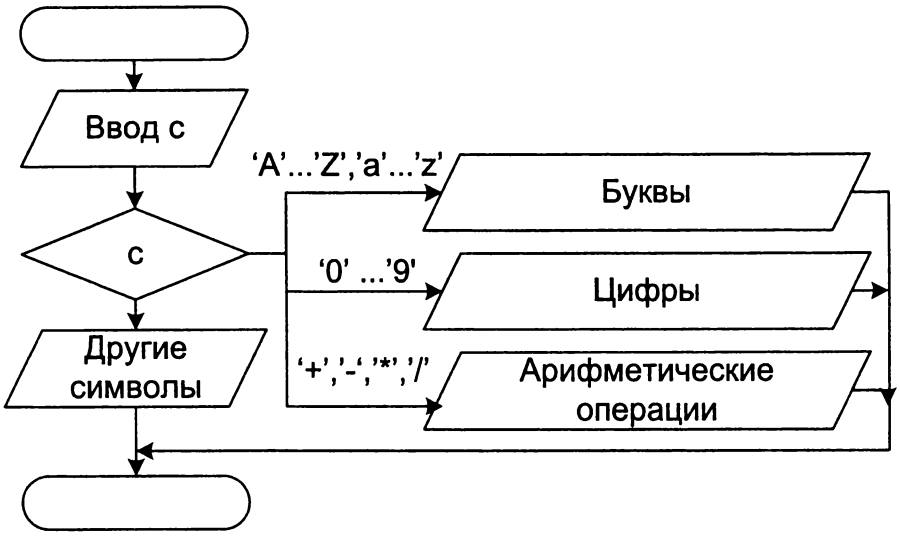
Результатом является вывод названия соответствующей группы.

Тестовый пример:

a) при c = '+' вывод «Арифметические операции»;

b) при c = '7' вывод «Цифры»;

c) при c = '$' вывод «Другие символы».



Контрольные вопросы

1. Какие типы данных могут являться селектором в операторе Case?

2. Если переменная *z* в условии задачи может принимать одно из четырех значений в зависимости от того, попадает ли *x* в интервал меньше 0, от 0 до корня квадратного от 2, от корня квадратного от 2 до 10 или больше 10, может ли эта задача быть решена с помощью оператора Case?

3. Какие из задач, предложенных в предыдущей лабораторной работе, можно решить с помощью оператора Case?

4. Определите, чему будет равно *S* после выполнения следующего оператора при $k = 0$:

```
Case k of
1: S:=0;
2.. 10: S:=1;
11: S:=3;
End;
```

5. Определите, чему будет равно *S* после выполнения следующего оператора при $k = 1$:

```
S:=1;
Case k of
1: S:=10;
0.. 10: S:=S+1;
11: S:=S+3
Else S:=0
End;
```

Задания для лабораторной работы

1. Некое предприятие ежесуточно расходует X кВт·ч электроэнергии – зимой, Y кВт·ч – летом и Z кВт·ч – весной и осенью. Составить программу для вычисления расхода электроэнергии *(R)* для заданного месяца текущего года.

2. Написать программу, которая будет выдавать все приходящиеся на введенный месяц праздничные дни.

3. Написать программу, которая по введенному номеру месяца будет выводить название сезона, к которому этот месяц относится, и проверять правильность введенного месяца.

4. Для натурального числа k, находящегося в интервале от 1 до 30000, вывести количество цифр в записи этого числа.

5. Ввести день, месяц и год, проверить правильность введенной даты и вывести дату следующего дня.

6. Составить программу дня определения знака Зодиака любого человека.

21 марта – 19 апреля – Овен;

20 апреля – 20 мая – Телец;

21 мая – 21 июня – Близнецы;

22 июня – 22 июля – Рак;

23 июля – 22 августа – Лев;

23 августа – 22 сентября – Дева;

23 сентября – 22 октября – Весы;

23 октября – 22 ноября – Скорпион;

23 ноября – 21 декабря – Стрелец;

22 декабря – 19 января – Козерог;

20 января – 18 февраля – Водолей;

19 февраля – 20 марта – Рыбы.

7. Ввести два положительных числа, меньшие 10. Сложить их и записать результат сложения этих чисел числительным.

8. Вычислить номер дня в невисокосном году по заданным числу и месяцу.

9. Для введенного числа k от 1 до 99 вывести слово «копеек» в нужной форме, т. е. с учетом значения k. Например: 20 копеек, но 32 копейки.

10. Вывести, какому животному восточного календаря соответствует введенный год. Годы внутри цикла носят следующие названия: Крысы, Быка, Тигра, Зайца, Дракона, Змеи, Лошади, Овцы, Обезьяны, Петуха, Собаки, Свиньи. Известно, что 2000 год – это год Дракона.

11. Составить программу, которая бы по вводимой дате выводила название дня недели, соответствующего этой дате.

12. Составить программу, которая по введенной дате рождения человека будет выводить, к какому цветку он относится.

1 января – 10 января – горечавка желтая;

11 января – 20 января – чертополох;

21 января – 31 января – бессмертник;

1 февраля – 10 февраля – омела;

11 февраля – 19 февраля – красавка;

20 февраля – 29 февраля – мимоза;

1 марта – 10 марта – мак;

11 марта – 20 марта – лилия;

21 марта – 31 марта – наперстянка;

1 апреля – 10 апреля – магнолия

11 апреля – 20 апреля – гортензия;

21 апреля – 30 апреля – георгин;

1 мая – 10 мая – ландыш;

11 мая – 21 мая – портулак;

22 мая – 31 мая – ромашка;

1 июня – 11 июня – колокольчик;

12 июня – 21 июня – маргаритка;

22 июня – 1 июля – тюльпан;

2 июля – 12 июля – кувшинка;

13 июля – 23 июля – фиалка;

24 июля – 2 августа – шиповник;

3 августа – 12 августа – подсолнух;

13 августа – 23 августа – роза;

24 августа – 2 сентября – дельфиниум;

3 сентября – 11 сентября – гвоздика;

12 сентября – 22 сентября – астра;

23 сентября – 3 октября – вереск;
4 октября – 13 октября – камелия;
14 октября – 23 октября – сирень;
24 октября – 2 ноября – фрезия;
3 ноября – 12 ноября – орхидея;
13 ноября – 22 ноября – пион;
23 ноября – 2 декабря – гладиолус;
3 декабря – 12 декабря – одуванчик;
13 декабря – 22 декабря – лотос;
23 декабря – 31 декабря – эдельвейс.

Тема 3. ЦИКЛИЧЕСКИЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС

Циклическим называется алгоритм, в котором некоторая часть операции (тело цикла, последовательность команд) выполняется многократно. Цикл не может быть бесконечным, так как алгоритм должен приводить к результату через конечное число шагов. В цикл также обязательно должен входить блок проверки условия. В зависимости от того, где располагается это условие, циклы делятся на циклы типа «пока» (условие располагается перед телом цикла) и циклы типа «до» (условие располагается после тела цикла).

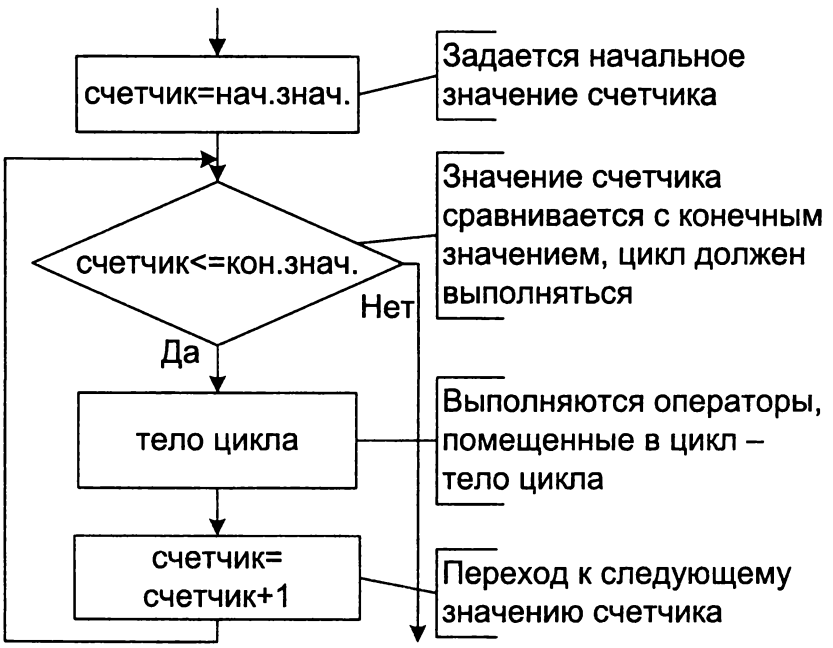
Другая классификация циклов – циклы со счетчиком и циклы с выходом по условию.

Лабораторная работа 4. Циклы со счетчиком

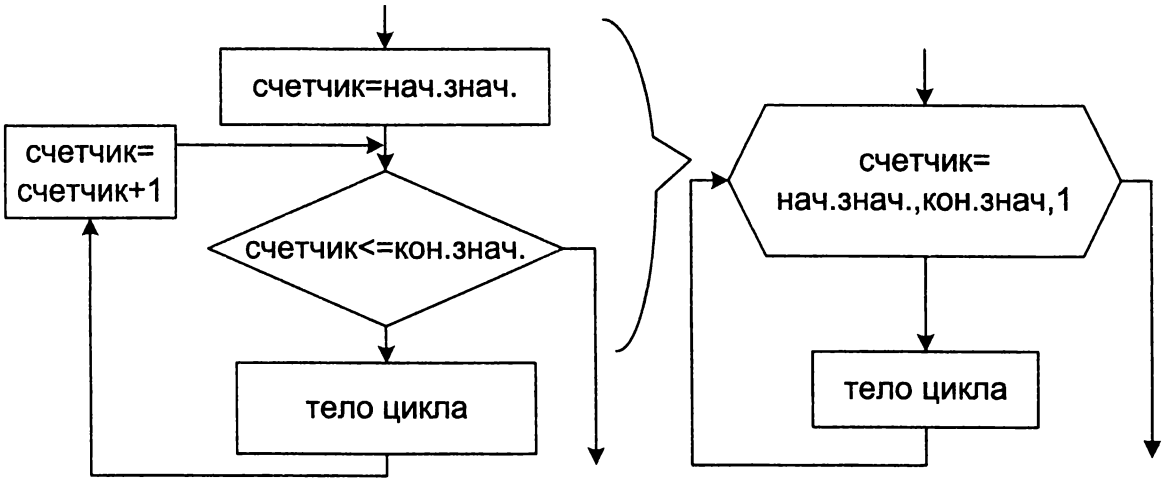
Теория

Цикл со счетчиком, или цикл с параметром, используется для организации циклов с фиксированным, заранее известным или определяемым во время работы программы числом повторений.

Алгоритм работы оператора цикла со счетчиком следующий:



В ГОСТ 19.701–90 [7] для сокращения принято еще одно обозначения цикла со счетчиком:



Оператор, соответствующий такой блок-схеме, имеет следующий вид:

For *счетчик:=нач.знач. to кон.знач. do*

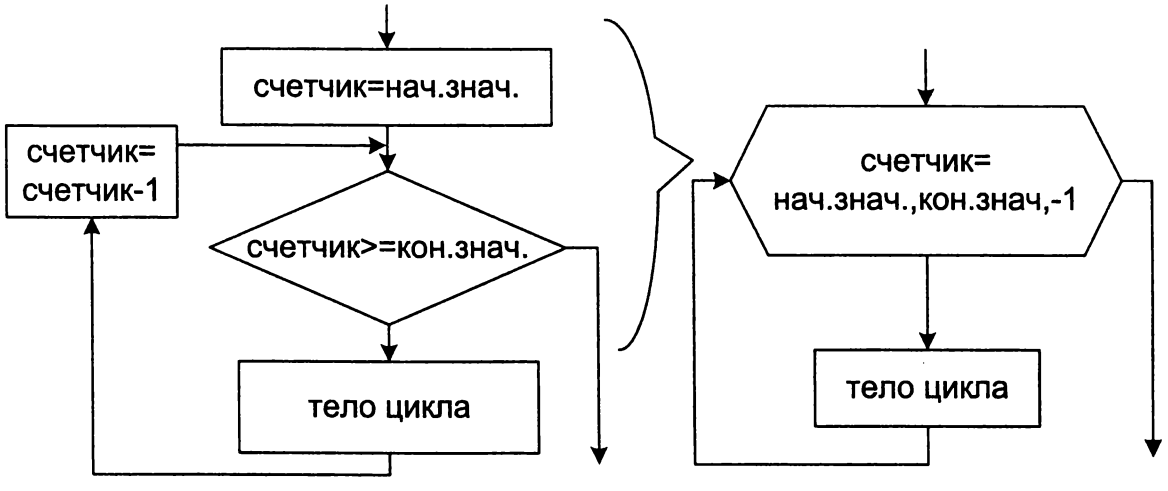
begin

тело цикла;

end;

Следует отметить, что в языке Pascal параметр цикла (счетчик) и все его значения могут иметь только порядковый тип данных. Приведенная блок-схема не полностью отражает возможности цикла *For*. Реально переход счетчика к следующему значению выполняется не увеличением счетчика на 1, а выполнением функции *Succ* («счетчик») – переход к следующему порядковому значению.

В цикле *For* счетчик может не только увеличиваться, но и уменьшаться. В этом случае цикл имеет следующий алгоритм:



Оператор, соответствующий такой блок-схеме, имеет следующий вид:

For *счетчик:=нач.знач. downto кон.знач. do*
begin
тело цикла;
end;

Примеры

Пример 1

Вычислить 10 значений d: $d = a^2 - b^2 + ab - 8$,

где $b = \begin{cases} c, & \text{если } c < -3, \\ \dfrac{c}{3}, & \text{если } -3 \leqslant c \leqslant 0, \\ a + c + 4, & \text{если } c > 0; \end{cases}$

a изменяется с шагом 4, начальное значение $a = 26$;

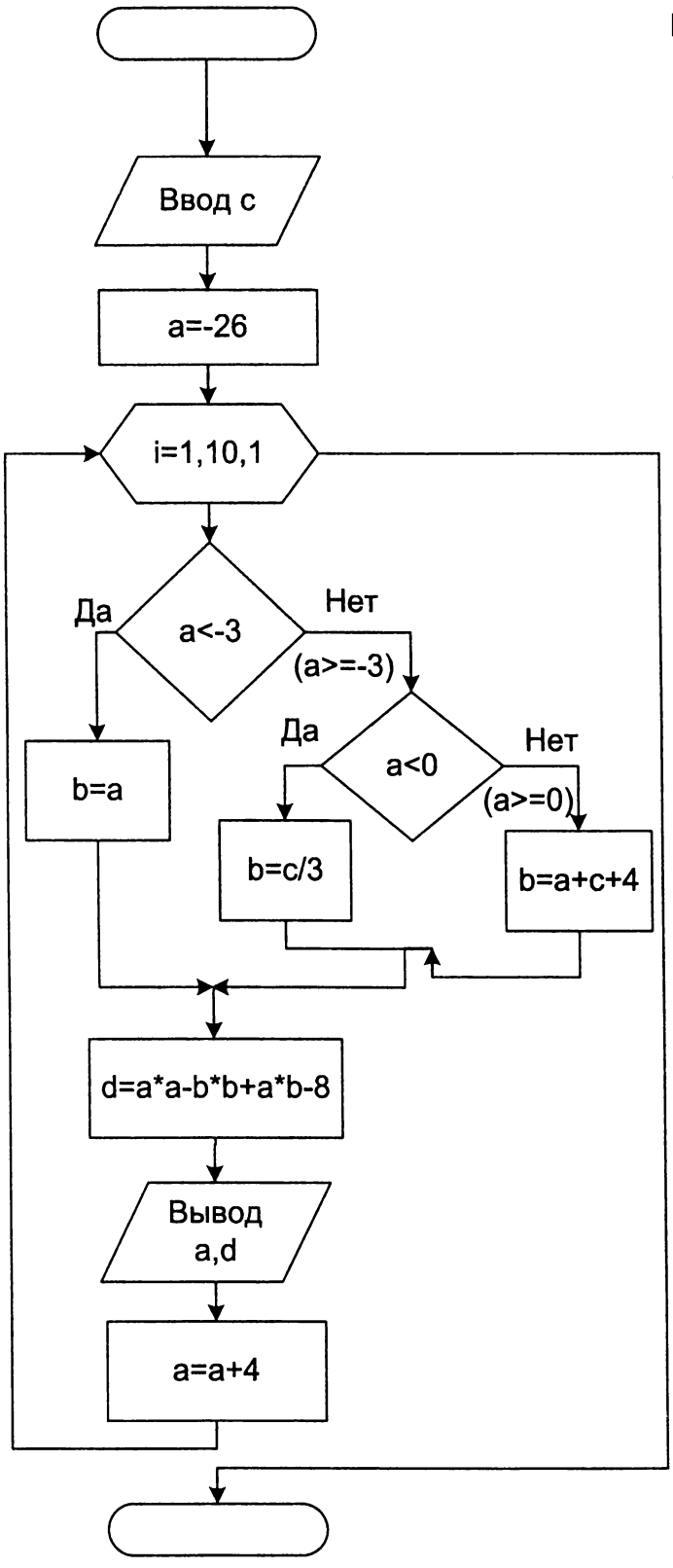
c (значение c) – произвольное число.

Исходные данные: значение c – вещественное число.

Результат: 10 значений d.

Тестовый пример: при $c = 3$.

i	a	d
1	–26	668
2	–22	476
3	–18	316
4	–14	188
5	–10	92
6	–6	28
7	–2	–7
8	2	–67
9	6	–63
10	10	–27




```
program cikl;
   var a,b,c,d: real;
       i: integer;

begin
    write('c='); readln(c);


    a:=-26;


for i:=1 to 10 do
      begin


      if a<-3 then b:=a
              else

           if a<0 then b:=c/3
                  else b:=a+c+4;
       d:=a*a-b*b-a*b-8;

       writeln('a= ',a:3:0, 'd=', d:5:1);

         a:=a+4;
       end;
```

end.

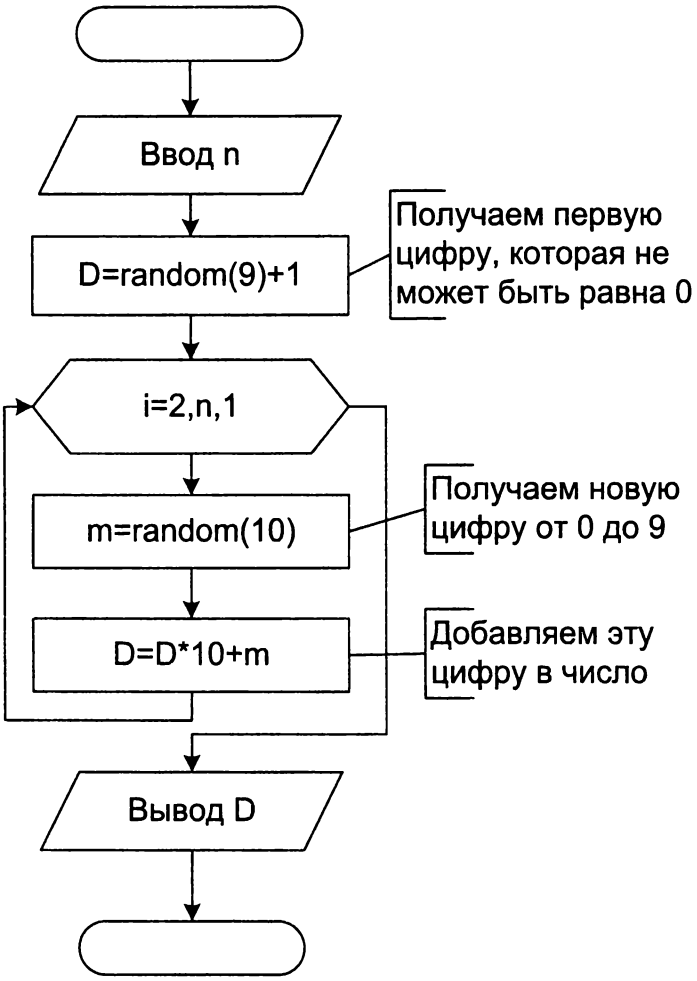
Пример 2

Дано натуральное $n < 10$. Получить n-значное натуральное число.

Исходные данные: n – количество цифр – целый тип.

Результат: D – целый тип.

Тестовый пример: проверяется только количество цифр.



Пример 3

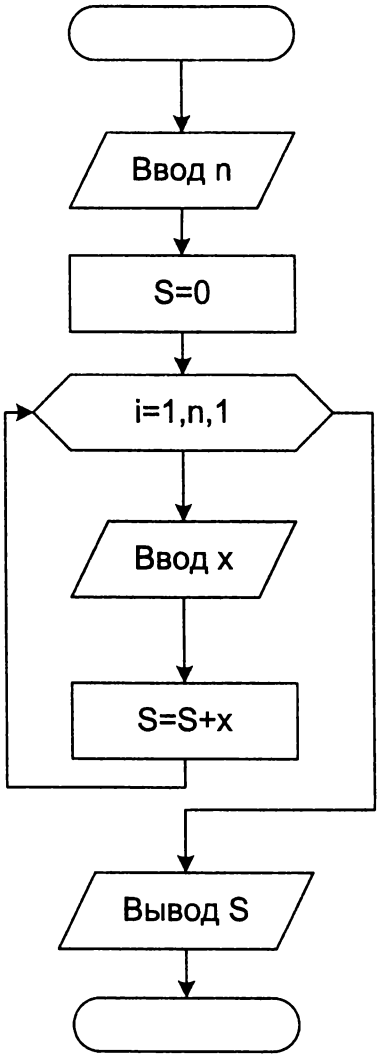
Вычислить сумму n вещественных чисел.

Исходные данные: n – количество введенных чисел – целый тип; x – переменная, куда помещаются вводимые числа – вещественный тип.

Результат: S – сумма введенных чисел – вещественный тип.

При вычислении суммы выполняется операция накопления данных в одной переменной, в данном случае – в переменной S: $S = S + x$. Эта операция должна выполняться n раз. Для того чтобы в результате первой операции накопления было получено значение первого x, переменную S предварительно надо очистить, т. е. $S = 0$.

Тестовый пример: при $n = 7$ и последовательности 1,2; 3; 5,6; 6,1; –4; –1; 4,2 $S = 17,1$.




```
program Sum;
 var n,i:integer;
     x,S:real;
begin
  write('n='); readln(n);
S:=0;
For i:=1 to n do
      begin
  write('Введите x ');
        readln(x);
S:=S+x;
   end;
writeln('S=',S:7:2);
end.
```

Пример 4

Во многих языках программирования отсутствует стандартная операция возведения в степень. Необходимо написать программу возведения в целую степень вещественного числа.

Исходные данные: n – степень, в которую возводится число, – целый тип; x – число, которое возводится в степень, – вещественный тип.

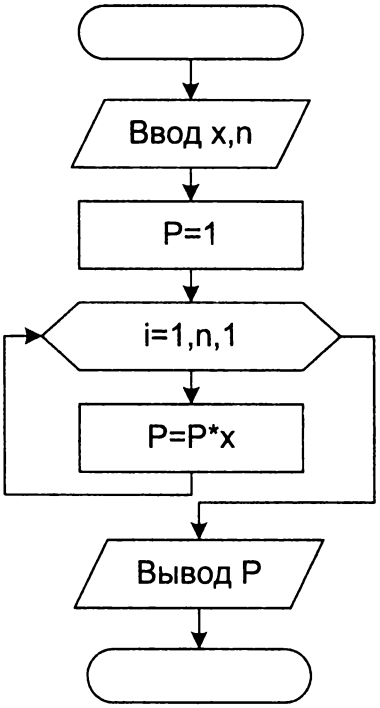
Результат: P – вещественный тип.

Возвести в n-ю степень число – значит, перемножить это число n раз: $P = x \cdot x \cdot x \cdot \ldots \cdot x$.

Операция перемножения x похожа на операцию сложения, только знак «+» заменяется на знак «*». Как и при сложении, в результате первого

действия умножения нужно получить само число x, т. е., и в этом отличие от суммирования, первоначально P должно быть равно 1.

Тестовый пример: при $x = 4$ и $n = 5$ $P = 1024$.



Пример 5

Ввести последовательность из n целых чисел. Найти минимальный член этой последовательности.

Исходные данные: n – количество введенных чисел – целый тип; a – переменная, куда помещаются вводимые числа, – целый тип.

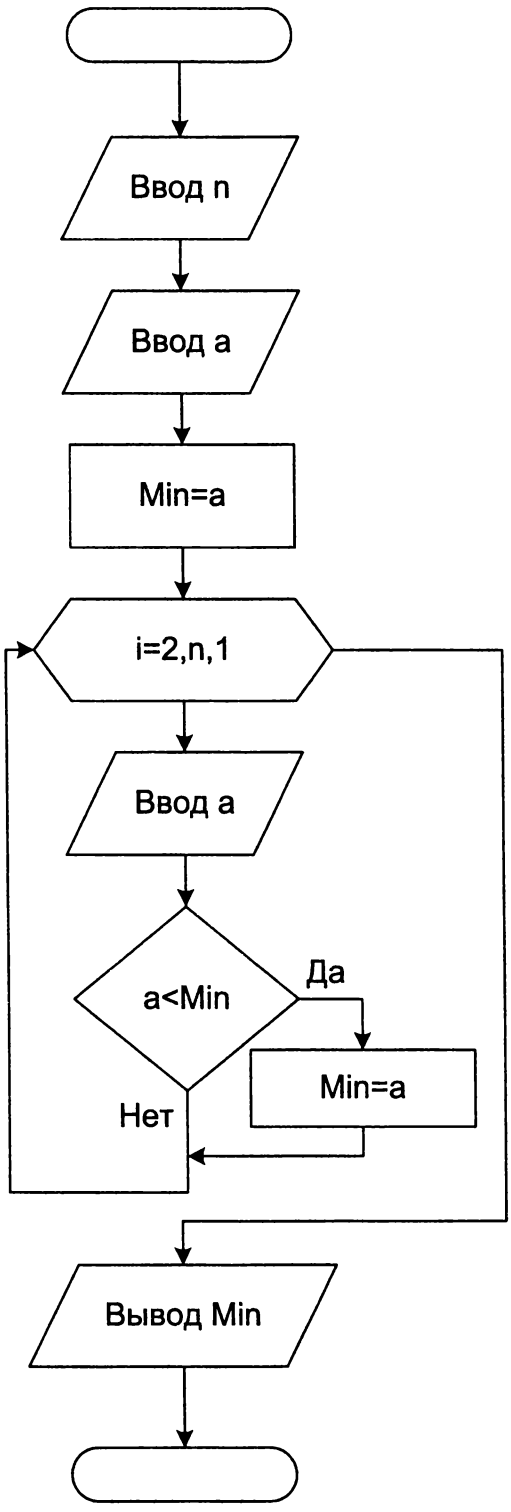
Результат: Min – целый тип.

Поиск минимального члена последовательности чисел похож на процедуру определения лучшего результата на соревновании лыжников с раздельным стартом. Когда первый лыжник прибегает на финиш, его результат считается лучшим, т. е. в Min записывается этот результат. Затем, когда прибегает очередной лыжник, его результат сравнивается с лучшим (Min), и если новый результат меньше, то в значение Min записывается уже он.

Следует обратить внимание, что первоначальное значение Min не может равняться нулю, так как в этом случае и в результате будет получен ноль.

Поиск максимального элемента отличается от поиска минимального только проверкой: если новое значение больше текущего максимального, в максимальное значение записывается это новое значение.

Тестовый пример: при $n = 7$ и последовательности 8, 0, –1, 2, 10, 3, 5 Min = –1.



Пример 6

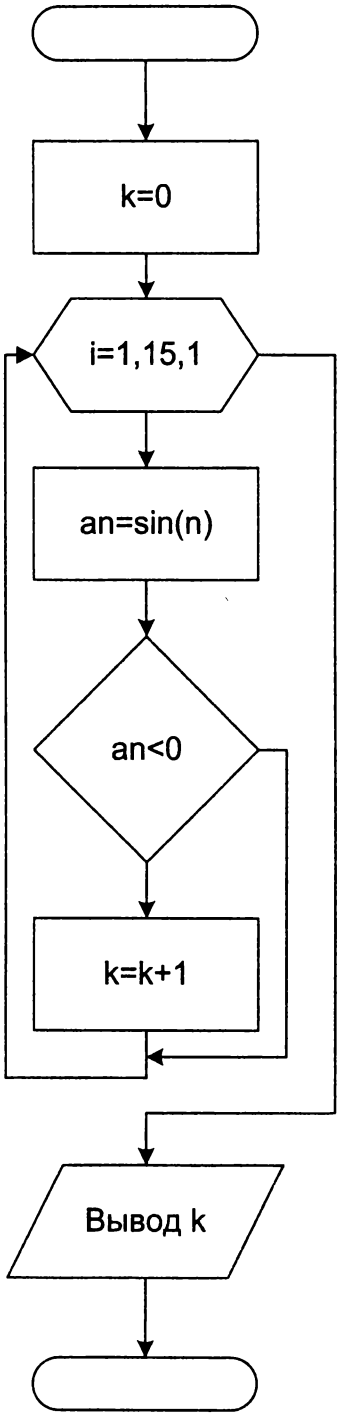
Дана числовая последовательность $a_n = \sin n$, где $n = 1, \ldots, 15$. Определить, сколько отрицательных элементов в этой последовательности.

Исходные данные: количество элементов последовательности – 15.

Результат: k – количество отрицательных элементов – целый тип.

Для нахождения количества отрицательных членов необходимо вычислить член последовательности и сравнить его с 0, если член меньше 0 – увеличить k на единицу. В начале решения задачи k необходимо обнулить.

Тестовый пример: с помощью программы *Excel* получаем $k = 6$.



Рассмотренные в примерах 1–6 алгоритмы относятся к так называемым стандартным алгоритмам и имеют много общего (табл. 4).

Таблица 4

Этапы выполнения стандартных алгоритмов

Этап	Алгоритм определения			
	суммы *(S)*	произведения *(P)*	количества (*k)*	минимума *(Min)*
До цикла	S:=0	P:=1	*k*:=0	*Min*:=первый элемент
1-й этап цикла	Получение элемента последовательности	Получение элемента последовательности	Получение элемента последовательности	Получение элемента последовательности
2-й этап цикла	Проверка того, надо ли данный элемент суммировать (необязательно)	Проверка того, надо ли данный элемент умножать (необязательно)	Проверка того, удовлетворяет ли элемент условию	Проверка того, как элемент соотносится с *Min*
3-й этап цикла	S:=S+элемент	P:=P*элемент	*k*:=*k*+1	*Min*:=элемент

Пример 7

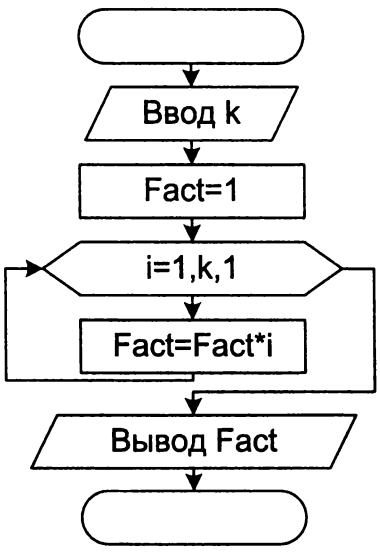
Вычислить факториал *k*. Факториалом называется произведение $k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k$.

Исходные данные: *k* – целый тип.

Результат: Fact – вещественный тип.

Факториал числа растет очень быстро. Факториал 8 уже равен 40320. Если программа написана для *Turbo Pascal*, эта величина больше, чем допустимое значение для типа *integer*. Поэтому результат должен иметь либо длинный целый тип, либо вещественный.

Тестовый пример: при *k* = 5 Fact = 120.



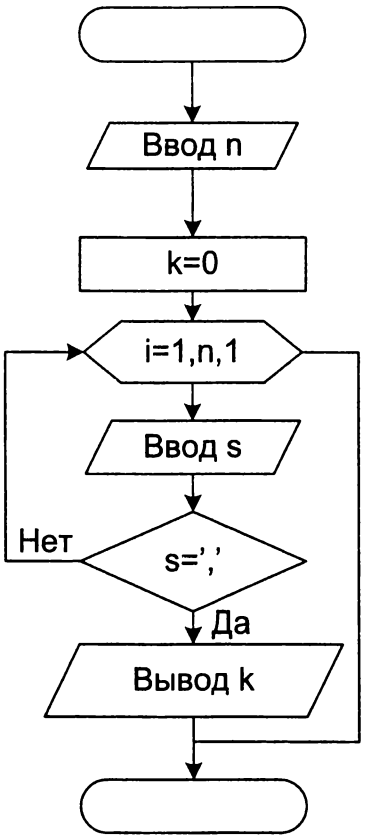
Пример 8

Даны натуральное число n, символы s_1, …, s_n. Известно, что среди s_1, …, s_n есть по крайней мере одна запятая. Найти такое натуральное i, что s_i – первая по порядку запятая.

Исходные данные: n – количество элементов – целое число, s – вводимый символ.

Результат: k – номер первой запятой.

Тестовый пример: при $n = 10$ и последовательности '4hgj, e,6+,' $k = 5$.




```
program simv;
 var n,k,i:integer;
      s:char;
begin
  write('n=');
   readln(n);
   k:=0;

writeln('Введите символы');
  For i:=1 to n do
    begin
      read(s);


      if s=',' then
              begin
                 writeln('номер запятой = ',k);
                 break;
              end;
            end;

end;
```

Во многих задачах требуется выполнять вычисления над элементами числовой последовательности, в которой каждый член – a_n – является функцией натурального аргумента – n. В примере 6 приведен вариант такой последовательности. Также по элементам числовой последовательности часто требуется определить формулу для вычисления n-го элемента.

Пример 9

Дана числовая последовательность $\{a_1 = 2, a_2 = 5, a_3 = 8, \ldots\}$. Члены последовательности с четными номерами заменили на обратные им числа

(например, 5 на –5). Найти сумму членов последовательности с десятого по тридцать первый включительно.

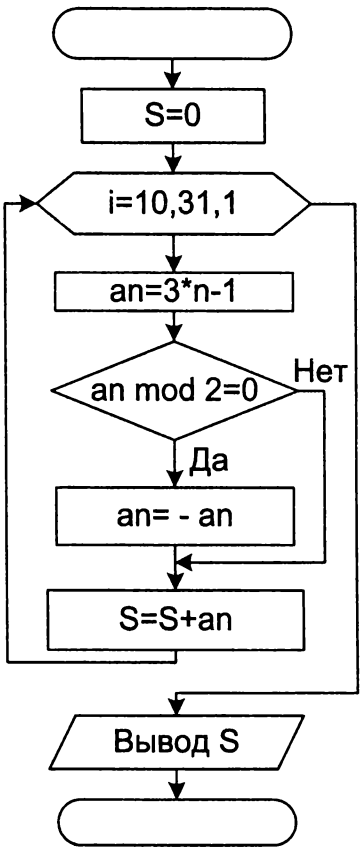
Исходные данные: начальный член последовательности – 10, конечный член последовательности – 31.

Результат: S – сумма членов последовательности – целый тип.

Необходимо определить формулу для вычисления члена последовательности в зависимости от его номера. С увеличением номера на 1 член последовательности изменяется на 3, т. е. член зависит от $3*n$. При $n = 1$ элемент равен 2. Чтобы получить 2, необходимо из $3*n$ вычесть 1. Таким образом, $a_n = (3*n) - 1$.

Чтобы определить четность номера, достаточно найти остаток от деления номера на 2. Если остаток равен 0, то номер четный.

Тестовый пример: с помощью программы *Excel* получаем $S = -33$.



Существуют числовые последовательности, в которых каждый новый член вычисляется через предыдущие. Формула, позволяющая сделать это, называется рекуррентной. То есть рекуррентная формула, если она существует, позволяет вычислить любой член последовательности посредством вычисления всех предыдущих членов.

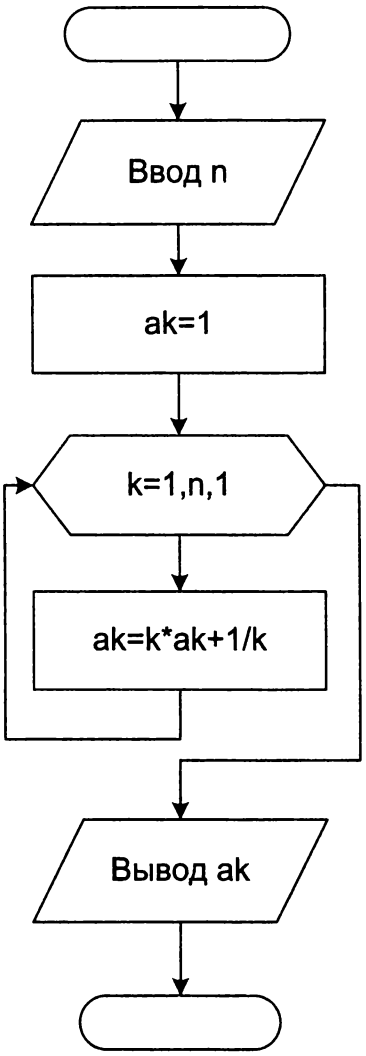
Пример 10

Пусть $a_0 = 1$; $a_k = ka_{k-1} + 1/k$, $k = 1, 2, \ldots$. Дано натуральное число n. Получить a_n.

Исходные данные: n – номер элемента – целый тип.

Результат: a – n-й элемент последовательности, – вещественный тип.

Тестовый пример: с помощью программы *Excel* получаем при $n = 5$ $a_n = 278{,}12$.



Пример 11

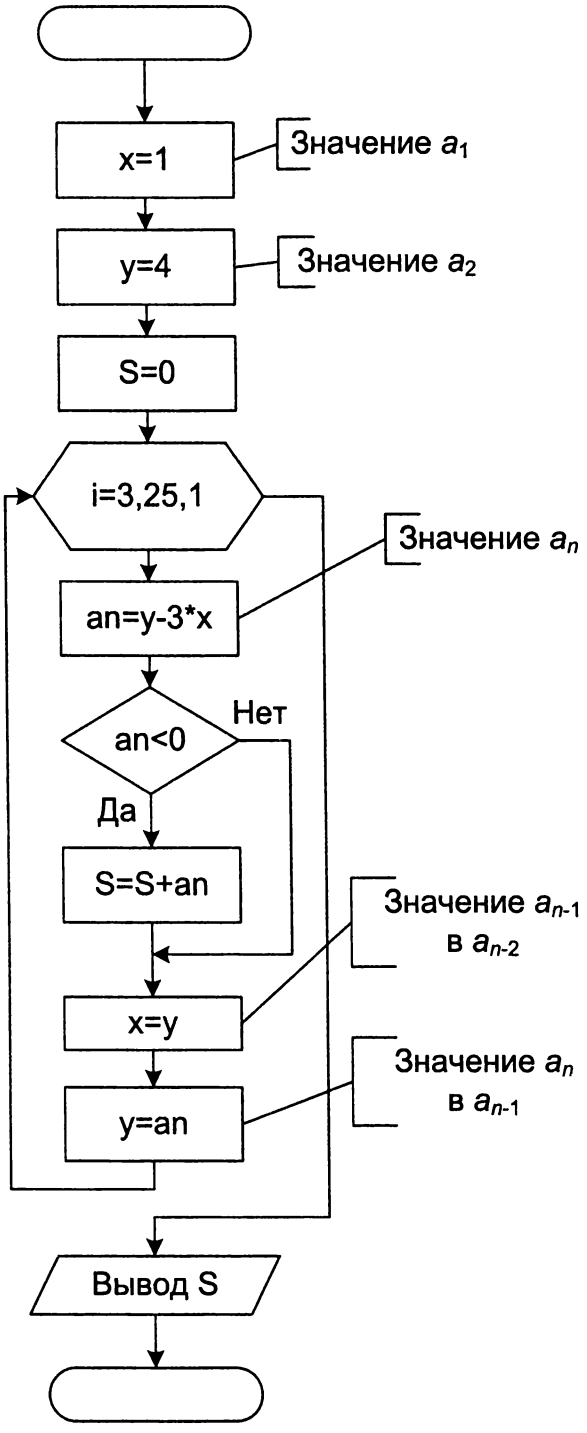
Дана числовая последовательность $\{a_1 = 0, a_2 = 4, \ldots, a_n = a_{n-1} - 3a_{n-2}\}$. Найти сумму отрицательных элементов последовательности при $n = 25$.

Исходные данные: n – количество элементов – целый тип, значение $x - a_{n-2}$ – целый тип, значение $y - a_{n-1}$ – целый тип.

Результат: S – сумма отрицательных элементов – целый тип.

Для вычисления нового значения a_n необходимо знать 2 предыдущих значения. В данном случае – значения a_{n-1} и a_{n-2}, которые необходимо сохранять в дополнительных переменных.

Тестовый пример: с помощью программы *Excel* получаем $S = -1644128$.



Пример 12

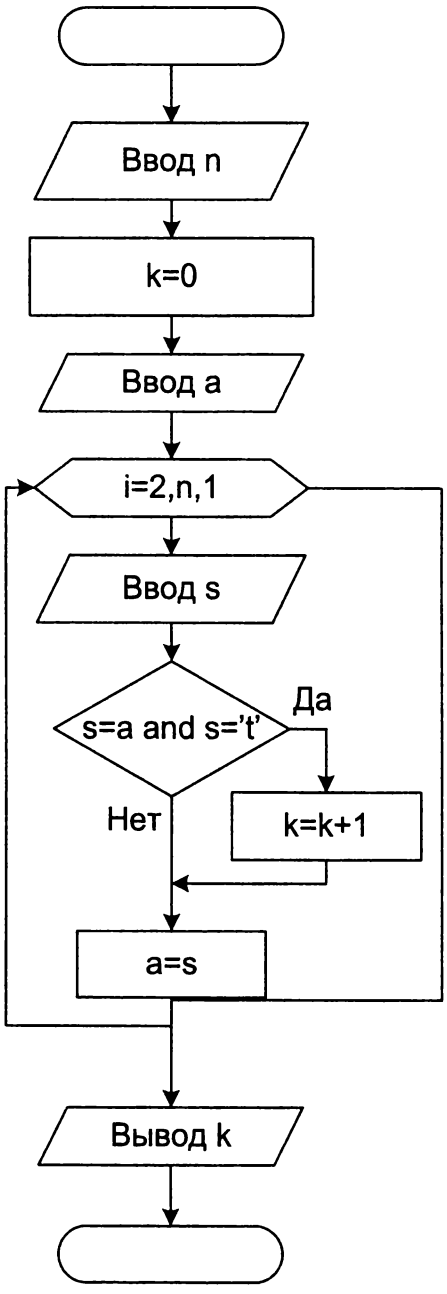
Даны натуральное число n, символы $s_1, \ldots, s_n$. Выяснить, имеются ли в последовательности $s_1, \ldots, s_n$ такие члены, что s_i и s_i+1 совпадают с t.

Исходные данные: n – количество элементов – целое число, s – вводимый символ.

Результат: k – количество пар, равных t.

Для сравнения пар введенных символов необходимо сохранять предыдущий введенный символ в переменной a.

Тестовый пример: при $n = 15$ в последовательности ‘ertbyttbnvttvcd’ $k = 2$.



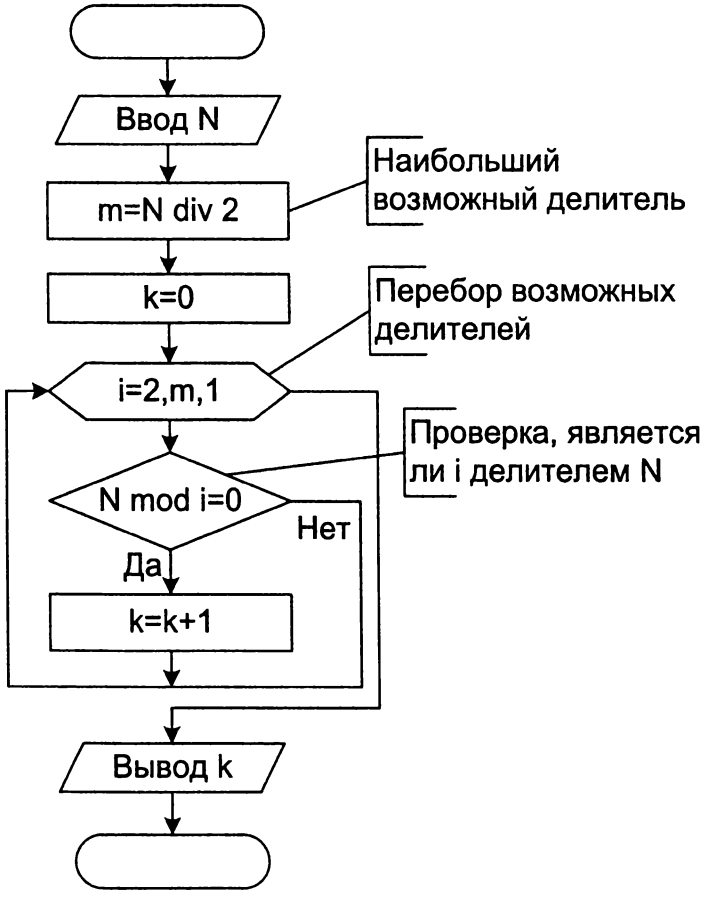
Пример 13

Дано натуральное число N. Определить, сколько у этого числа делителей (1 и само число не учитывать).

Исходные данные: N – целый тип.

Результат: k – количество делителей – целый тип.

Тестовый пример: при $N = 24$ $k = 6$.



Пример 14

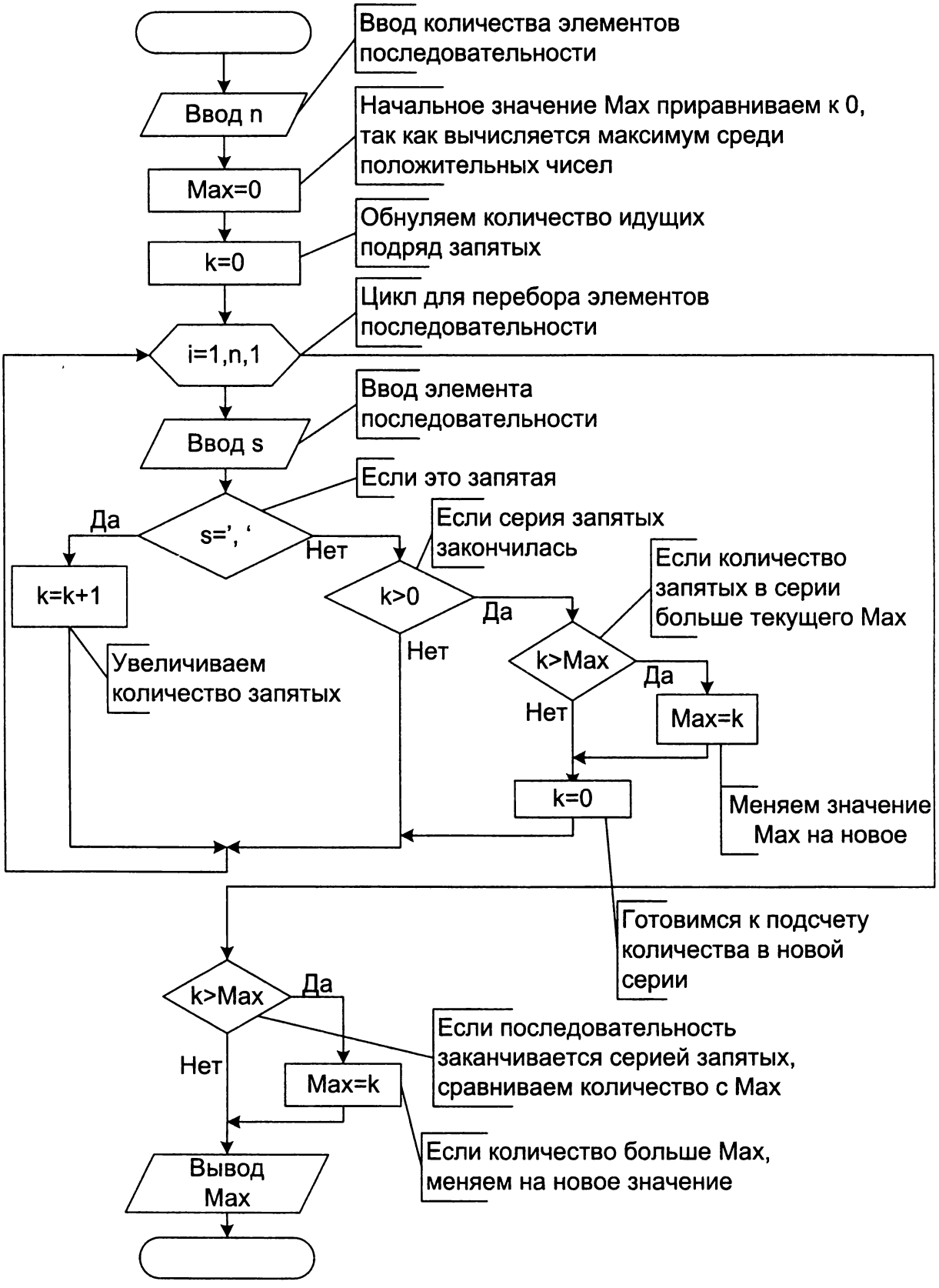
Даны натуральное число n, символы s_1, …, s_n. Подсчитать наибольшее количество идущих подряд запятых.

Исходные данные: n – натуральное число, s – вводимые символы.

Результат: Max – максимальное количество запятых – целый тип.

Если введенный символ равен «,», нужно увеличивать количество запятых, идущих подряд, на 1 (k). Если символ не запятая, следует проверить k; если символ следует сразу же после запятой ($k > 0$), надо сравнить k с максимальным количеством запятых на данный момент. Если $k >$ Max – изменить значение Max. После сравнения необходимо обнулить k перед поиском количества запятых в новой серии запятых.

Тестовый пример: при n = 20 и последовательности ’23,, wer,,,,, tyu,,,’ Max = 5.


Ввод количества элементов последовательности
Ввод n
Начальное значение Max приравниваем к 0, так как вычисляется максимум среди положительных чисел
Max=0
Обнуляем количество идущих подряд запятых
k=0
Цикл для перебора элементов последовательности
i=1,n,1
Ввод элемента последовательности
Ввод s
Если это запятая
s=', '
Да
Нет
k=k+1
Увеличиваем количество запятых
Если серия запятых закончилась
k>0
Да
Нет
Если количество запятых в серии больше текущего Max
k>Max
Да
Нет
Max=k
k=0
Меняем значение Max на новое
Готовимся к подсчету количества в новой серии
k>Max
Да
Нет
Если последовательность заканчивается серией запятых, сравниваем количество с Max
Max=k
Если количество больше Max, меняем на новое значение
Вывод Max

Контрольные вопросы

1. Какой тип данных может иметь счетчик цикла?

2. Может ли быть у счетчика цикла шаг, равный 2?

3. Как можно выйти из цикла досрочно?

4. Если начальное значение счетчика окажется меньше конечного, будет ли тело цикла выполняться хотя бы один раз?

5. Можно ли при поиске максимального значения в произвольной последовательности чисел первоначальное значение максимума задать равным нулю? Почему?

6. Как будет выглядеть блок-схема примера 5, если потребуется найти не минимум, а максимум?

Задания для лабораторной работы

1. Спортсмен в первый день пробежал 10 км. В каждый последующий день он пробегал на 10 % больше, чем в предыдущий. Найти, какой путь пробежит спортсмен на 7-й день.

2. Даны натуральное число n, символы $s_1, \ldots, s_n$. Подсчитать:

• сколько раз среди данных символов встречается символ «+» и сколько раз – символ «*»;

• общее число символов «+»,«-», «*» в последовательности $s_1, \ldots, s_n$.

3. Даны натуральное число n, символы $s_1, \ldots, s_n$. Известно, что среди символов $s_1, \ldots, s_n$ есть по крайней мере одна запятая. Найти такое натуральное i, при котором s_i – последняя по порядку запятая.

4. Дана числовая последовательность $\frac{1}{8}; \frac{1}{12}; \frac{1}{16}; \ldots$. Найти сумму ее членов с 10-го по 25-й включительно.

5. Дано натуральное число n. Найти наибольшее среди чисел (k = 1, …, n), а также сумму всех этих чисел.

6. Вычислить $\sum_{i=1}^{30}(a_i - b_i)^2$, где $a_i = \begin{cases} i, & \text{если } i - \text{нечетное}, \\ i/2 & \text{в противном случае} \end{cases}$,

$b_i = \begin{cases} i^2, \text{если } i - \text{нечетное}, \\ i^3 \text{ в противном случае.} \end{cases}$

7. Даны натуральное число n, действительные числа $y_1, y_2, \ldots, y_n$. Найти Max ($|z_1|, |z_2|, \ldots, |z_n|$), где $z_i = \begin{cases} y_i \text{ при } |y_i| \le 2, \\ 0{,}5 \text{ в противном случае.} \end{cases}$

8. Даны натуральное число n, действительные числа a_1, …, a_n. Определить в этой последовательности число соседств двух чисел с разными знаками.

9. Пусть $x_1 = 0{,}3$; $x_2 = -0{,}3$; $x_i = i + \sin x_{i-2}$, $i = 3, 4, \ldots, 100$. Найти в этой последовательности значение, ближайшее к какому-нибудь целому.

10. Даны натуральное число n, символы s_1, …, s_n. Выяснить, имеются ли в последовательности s_1, …, s_n такие члены последовательности s_i, $s_i + 1$, что s_i – это запятая, а $s_i + 1$ – тире.

11. Даны натуральное число n, символы s_1, …, s_n. Выяснить, верно ли, что в последовательности s_1, …, s_n имеются пять идущих подряд букв е.

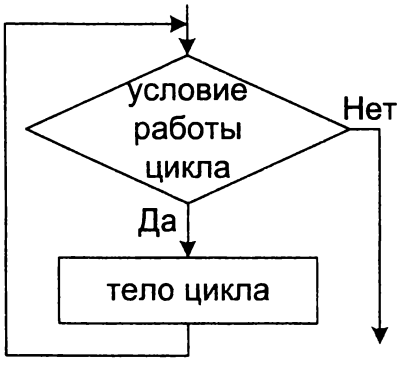
12. Даны натуральное число n, символы s_1, …, s_n. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Подсчитать количество букв в последнем слове данной последовательности.

Лабораторная работа 5. Цикл с условием

Теория

Когда неизвестно, сколько раз сработает цикл, используются ***циклы с условием***: окончание цикла происходит после выполнения / невыполнения условия. Существуют два таких оператора цикла: *цикл с предусловием*, или цикл «пока», и *цикл с постусловием*, или цикл «до».

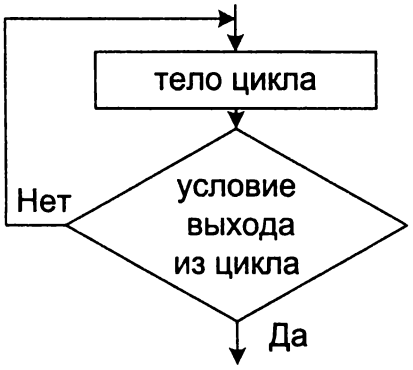
Цикл с предусловием сначала проверяет, нужно ли выполнять операторы в цикле (*тело цикла*), а затем выполняет эти операторы. Блок-схема такого цикла и реализующий ее оператор выглядят следующим образом:




```
while условие работы цикла
do

begin
тело цикла
end;
```

Точно так же, как и в операторе со счетчиком, если тело цикла состоит из нескольких операторов, используются операторные скобки *begin* и *end*. Если тело цикла состоит из одного оператора, операторные скобки можно не использовать.

В цикле с постусловием сначала выполняется тело цикла, а потом идет проверка условия выхода из цикла. Блок-схема и оператор в этом случае имеют следующий вид:




```
repeat
тело цикла

until условие выхода
из цикла;
```

Любой оператор цикла *For* может быть заменен оператором цикла while. Но возможности оператора цикла *while* больше:

- счетчик является действительным числом;
- шаг изменения параметра цикла не целое число;
- шаг может изменяться в цикле;
- конечное значение изменяется в цикле.

Выбирать оператора *repeat* следует с осторожностью, так как у него тело цикла обязательно выполняется хотя бы один раз. Поэтому оператор *repeat* используется для проверки правильности ввода, когда условие выхода определяется внутри цикла или когда тело цикла должно выполняться хотя бы один раз.

Оператор досрочного выхода из цикла – *break*.

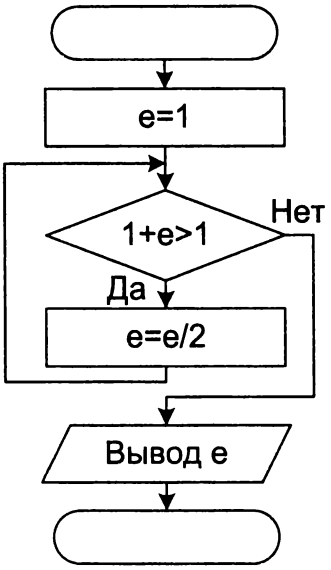
Примеры

Пример 1

Вещественные числа в программировании всегда являются приближенными, т. е. несут в себе погрешность, которая называется погрешностью машинного округления. Разность между вещественной единицей и ближайшим к ней числом, представимым в памяти компьютера, называется машинным эпсилон (ε). Найти машинный ε.

Исходные данные: начальное значение $e = 1$, e – вещественное число.

Результат: значение, которое существенно для компьютера.




```
program epsilon;
 var e:real;
begin
    e:=1;

while 1+e>1 do
   e:=e/2;

writeln('epsilon',e);

end.
```

Пример 2

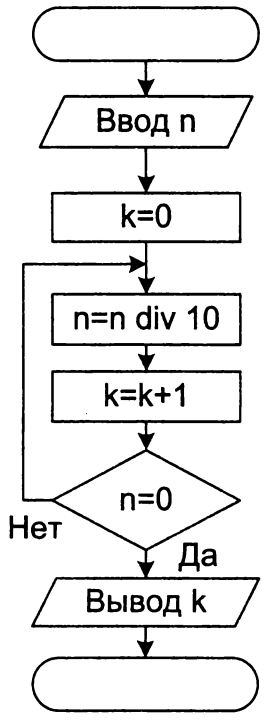
Дано натуральное число. Определить, сколько цифр в этом числе, воспользовавшись циклом с постусловием.

Исходные данные: n – целый тип.

Результат: k – количество цифр – целое.

Для получения количества цифр следует выполнять целочисленное деление на 10 до достижения 0.

Тестовый пример: при $n = 12345$ $k = 5$.




```
program kol_n;
  var n,k:integer;
 begin
   write('n=');
readln(n);
    k:=0;
    repeat
    n:=n div 10;
    pk:=k+1;
    until n=0;
    writeln('k= ',k)
end.
```

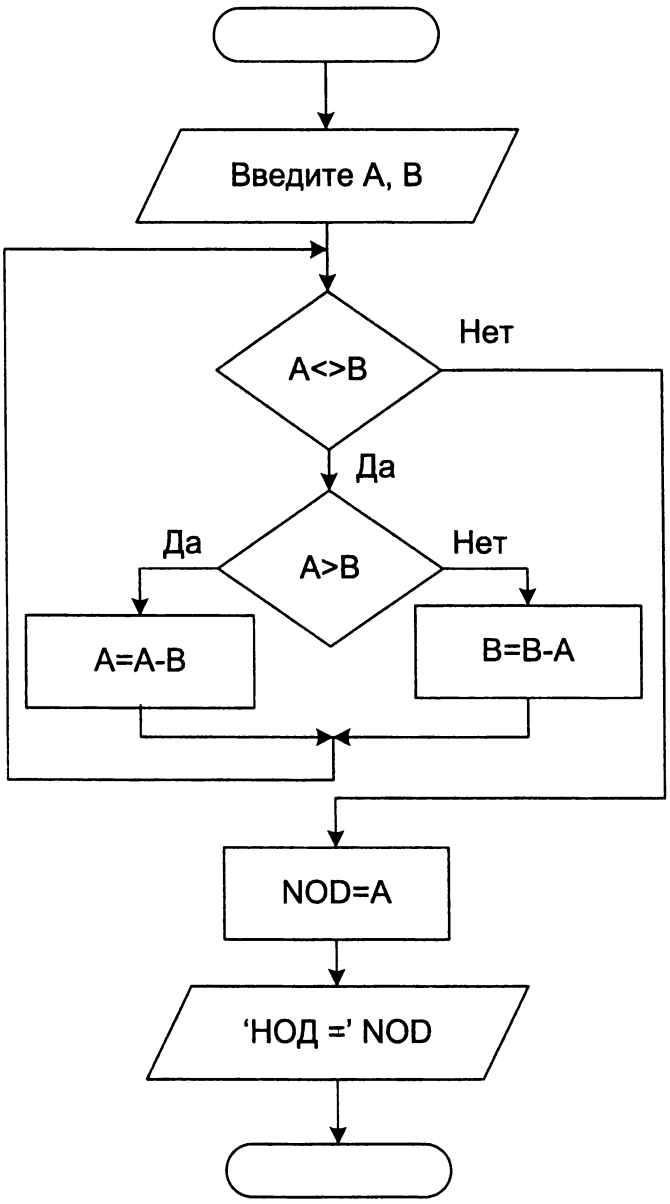
Пример 3

Даны целые числа A и B. Найти наибольший общий делитель этих чисел, воспользовавшись алгоритмом Евклида.

Исходные данные: A, B – целый тип.

Результат: NOD (НОД) – наибольший общий делитель.

Тестовый пример: при $A = 36$, $B = 48$ НОД = 12.



Пример 4

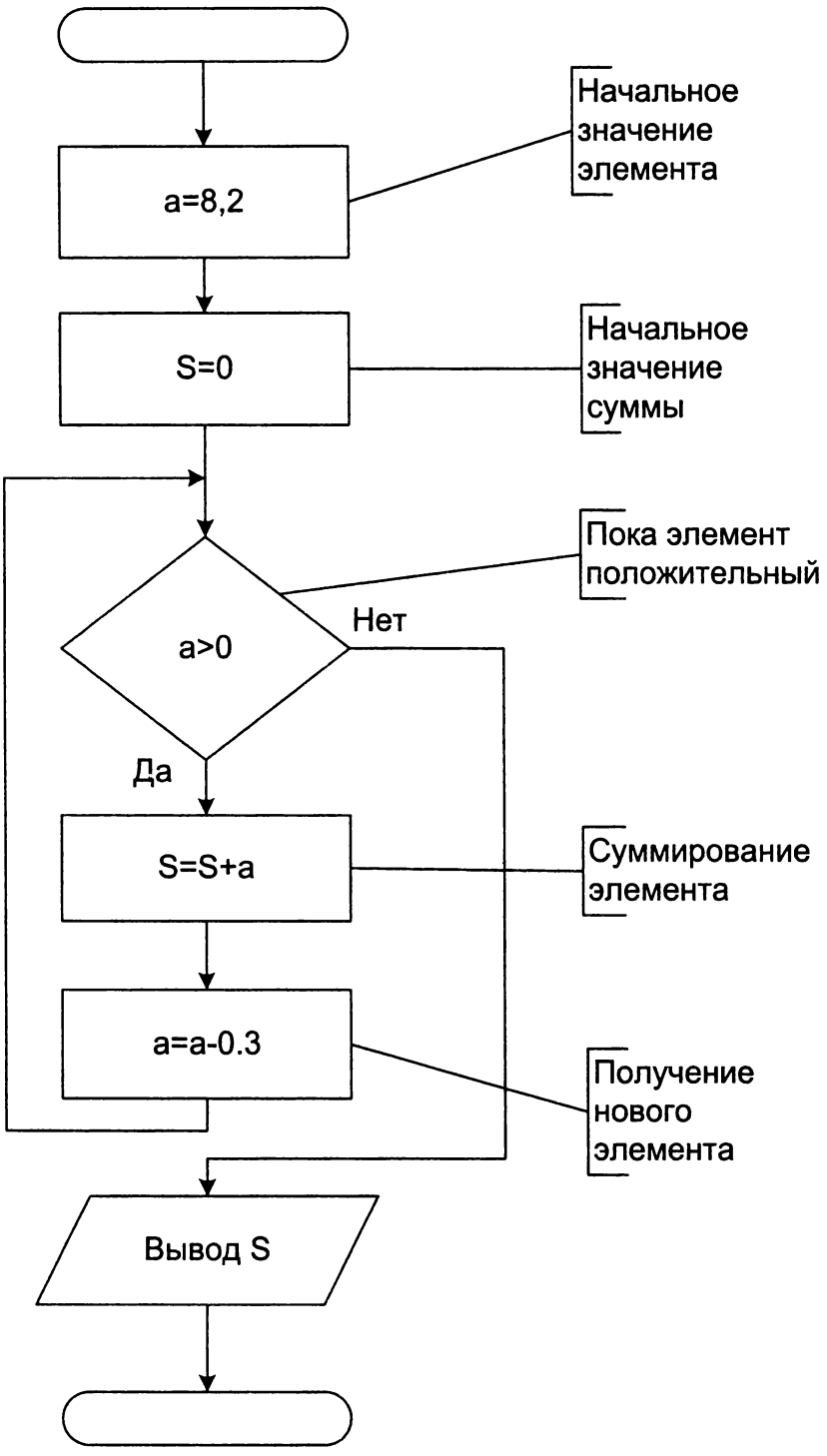
Дана числовая последовательность $\{a_1 = 8{,}2, a_2 = 7{,}9, a_3 = 7{,}6, \ldots\}$.

Найти сумму всех положительных элементов этой последовательности.

Исходные данные: a – элементы последовательности – вещественный тип.

Результат: S – сумма положительных элементов последовательности – вещественный тип.

Тестовый пример: $S = 116{,}2$.



Пример 5

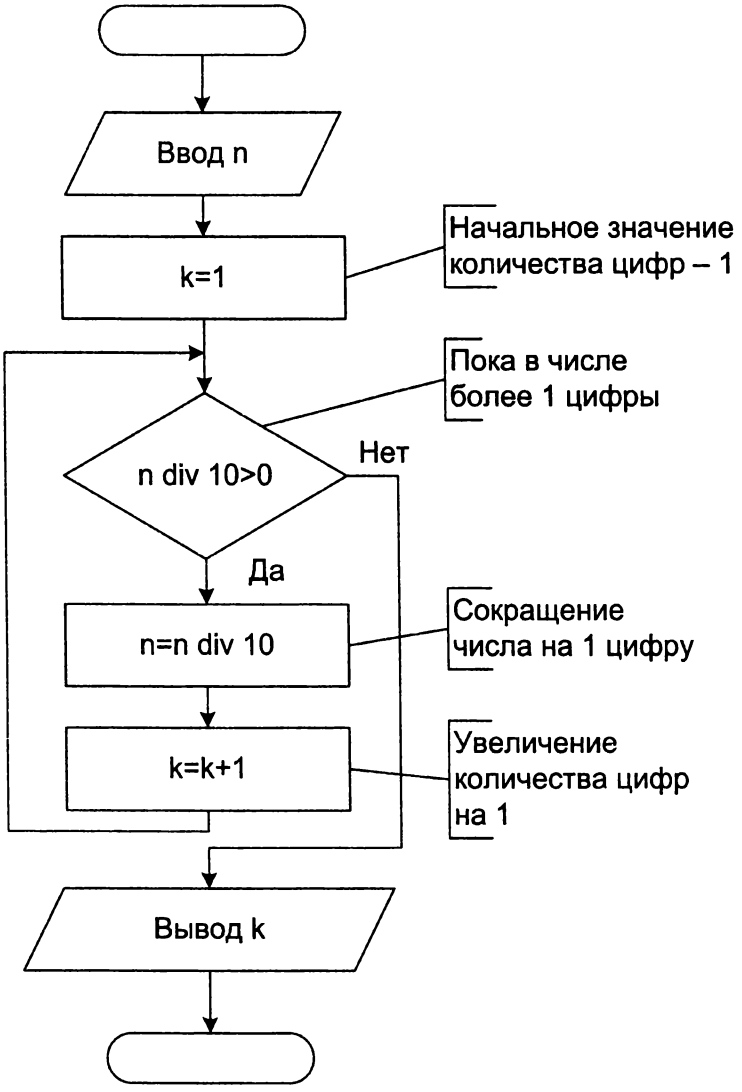
Дано натуральное число. Определить, сколько цифр в этом числе, воспользовавшись циклом с предусловием.

Исходные данные: n – целый тип.

Результат: k – количество цифр – целый тип.

Для получения количества цифр следует выполнять целочисленное деление на 10 до достижения 0.

Тестовый пример: при $n = 12345$ $k = 5$.



Пример 6

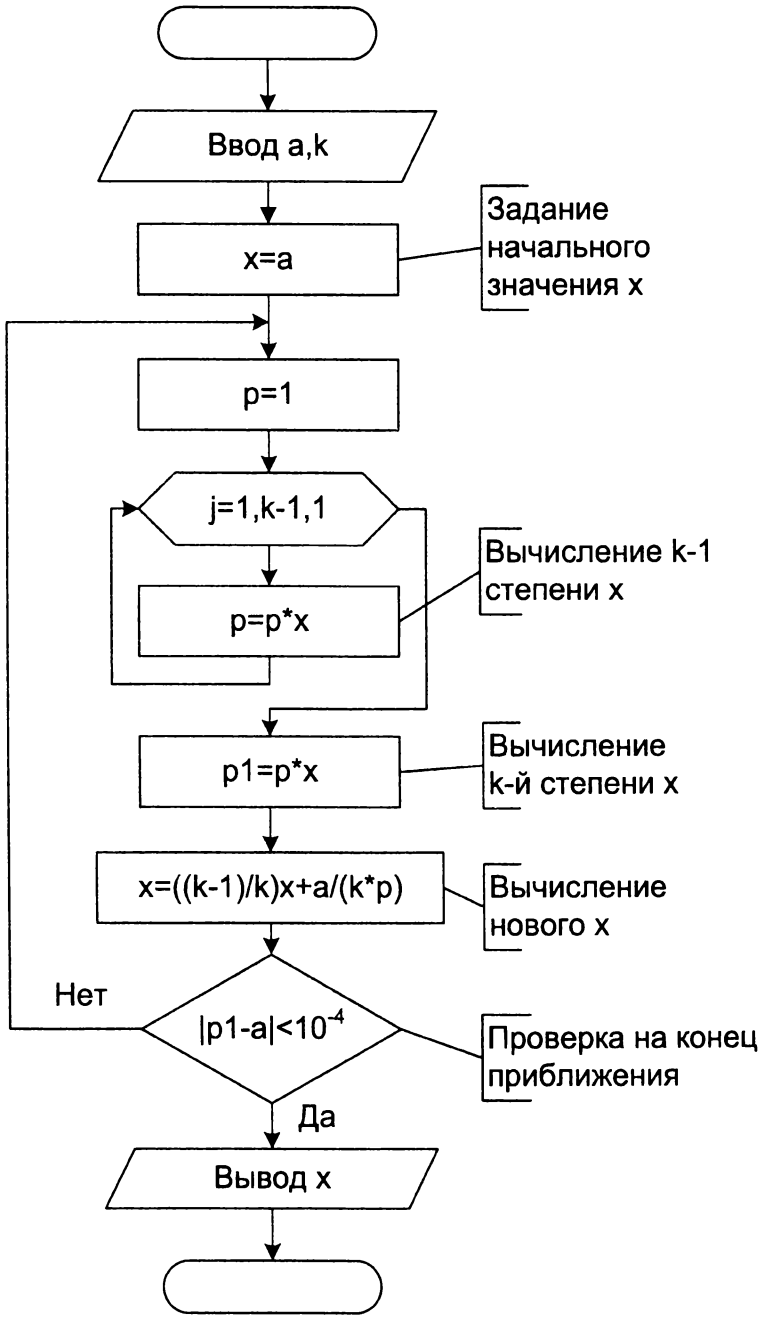
Создать программу для вычисления значения выражения $\sqrt[k]{a}$.

Для этого необходимо вычислить элементы числовой последовательности $x_0 = a$: $x_i = \frac{k-1}{k} x_{i-1} + \frac{a}{k x_{i-1}^{k-1}}$, $i = 1, 2, \ldots$ и найти первое значение x_n, для которого $|x_n^k - a| < 10^{-4}$.

Исходные данные: a – вещественный тип, k – целочисленный тип.

Результат: x – вещественный тип.

Тестовый пример: при $a = 8$, $k = 3$, $x = 2$.



Пример 7

Дано натуральное число N. Установить, верно ли утверждение, что цифры в этом числе образуют возрастающую последовательность.

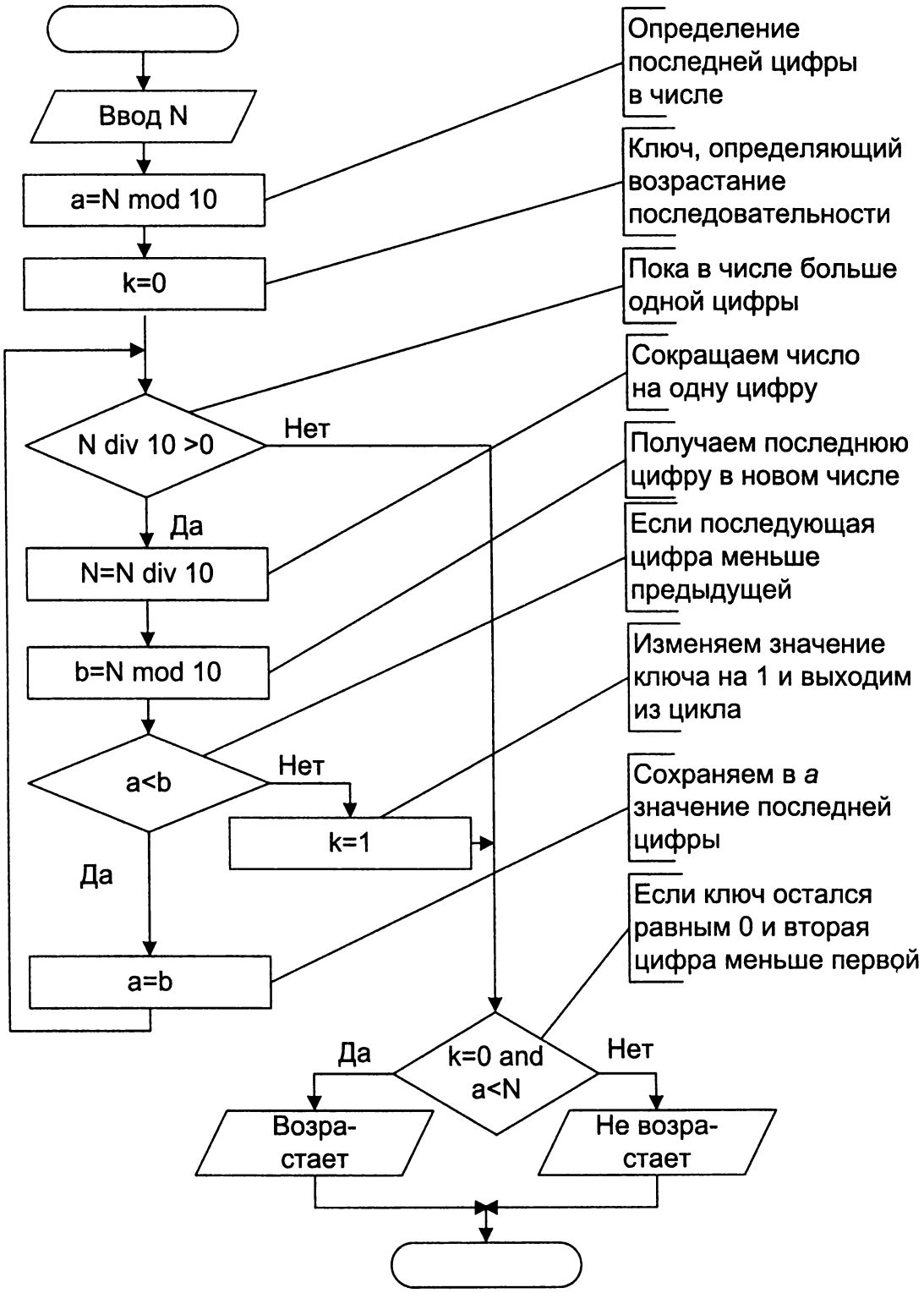
Исходные данные: N – целый тип.

Результат: ключ k равен нулю, если последовательность цифр возрастает, и единице, если не возрастает.

Для сокращения числа на одну цифру делим число на 10, для получения цифр в числе вычисляем остаток от деления на 10. Чтобы сравнивать

две цифры, используем переменные a и b: a – правая цифра в паре, b – левая цифра в паре.

Тестовый пример: при $N = 35679$ вывод «Возрастает»; при $N = 35479$ вывод «Не возрастает».



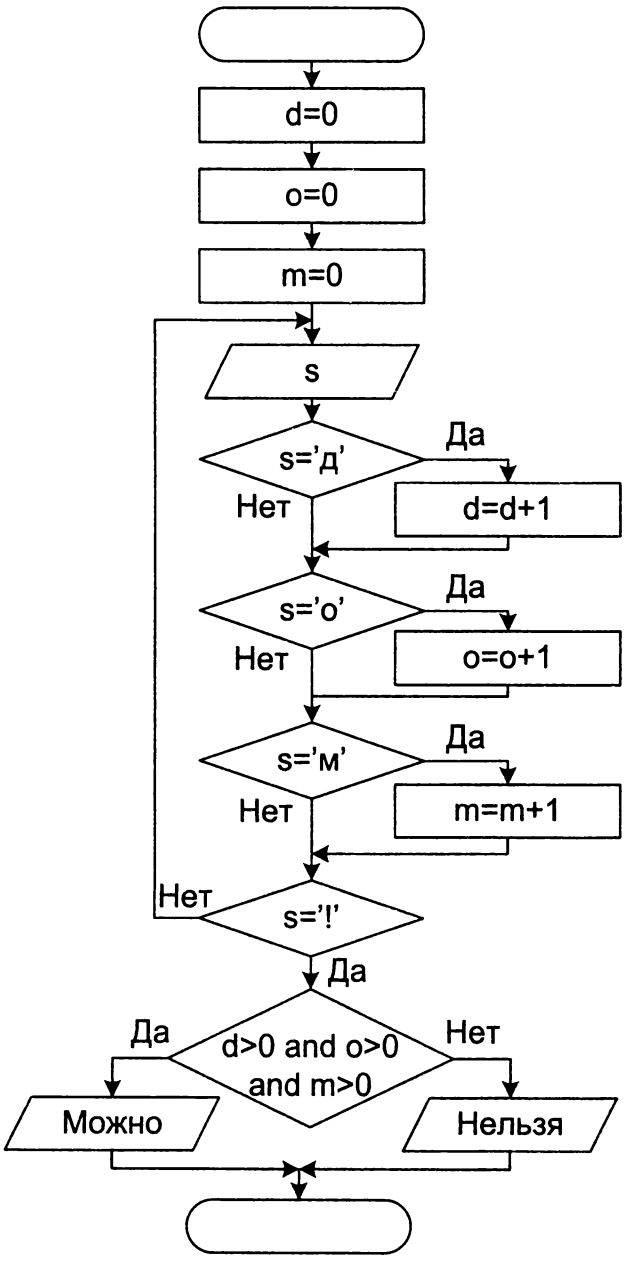
Пример 8

Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, … есть по крайней мере один восклицательный знак. Пусть s – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, верно ли, что среди символов имеются все буквы, входящие в слово «дом».

Исходные данные: s – элементы последовательности – символьный тип.

Результат: о – количество букв «о» – целый тип, d – количество букв «д» – целый тип, m – количество букв «м» – целый тип. Если каждое из этих значений больше 0 – из символов последовательности можно составить слово «дом».

Тестовый пример: при вводе последовательности «имркпооовдя!» вывод «Можно», при вводе последовательности «имркпгрпоовя!» вывод «Нельзя».



Контрольные вопросы

1. Когда следует использовать цикл «до», а когда цикл «пока»?

2. К какому виду цикла относится цикл со счетчиком?

3. Какой из трех рассмотренных циклов является универсальным?

4. Напишите приведенный фрагмент через цикл *while*:

For *i*:=1 to 8 do $x = x+i$;

5. При каких условиях вместо цикла со счетчиком приходится использовать цикл *while*?

6. Всегда ли цикл *while* можно заменить циклом *repeat*?

7. Какой оператор следует использовать, если необходимо досрочно выйти из цикла?

Задания для лабораторной работы

1. Спортсмен в первый день пробежал 10 км. В каждый последующий день он пробегал на 10 % больше, чем в предыдущий. Через сколько дней спортсмен в сумме пробежит 100 км?

2. Сколько чисел нужно взять в последовательности $\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots$, чтобы получить число, большее, чем N?

3. Шары расположены в форме треугольника так, что в первом ряду находится один шар, во втором – два, в третьем – три и так далее. Сколько рядов удастся построить, если имеется N шаров?

4. Вывести значения K, для которых Z = U + M – 20K + T больше 0, если

$$K = \begin{cases} 2U, \text{ если } M < 2, \\ U, \text{ если } 2 \leqslant M \leqslant 4 \text{ и } U = 3M + 1, \\ 1 - M, \text{ если } M > 4; \end{cases}$$

M изменяется от 0 до 6 с шагом 1,5; T – произвольное число.

5. Дано натуральное число n. Выяснить, входит ли цифра 3 в запись числа $n2$.

6. Дано натуральное число n. Чему равна сумма его цифр?

7. Даны действительные числа x, y ($x > 1$, $y > 1$). Получить целое число k, удовлетворяющее условию $y^{k-1} \leqslant x < y^k$.

8. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, … есть по крайней мере один восклицательный знак. Пусть s_2, s_3, …, s_n – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Определить количество пробелов среди s_1, …, s_n.

9. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, ... есть по крайней мере один восклицательный знак. Пусть s_2, s_3, ..., s_n – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, имеется ли среди s_1, ..., s_n пара соседствующих одинаковых символов.

10. Даны натуральное число n, символы. Известно, что символ s_1 отличен от восклицательного знака и что среди s_2, s_3, ... есть по крайней мере один восклицательный знак. Пусть s_2, s_3, ..., s_n – символы данной последовательности, предшествующие первому восклицательному знаку (n заранее неизвестно). Выяснить, верно ли, что в последовательности имеются пять идущих подряд букв «е».

11. Дано натуральное число N. Если оно содержит пять цифр, то получить новое число M, которое образуется путем исключения средней цифры исходного числа. Если количество цифр не пять, то $M = N$.

12. Среди всех n-значных чисел указать те, сумма цифр которых равна заданному числу k.

Лабораторная работа 6. Вычисления с точностью

Теория

Число x называется ***пределом числовой последовательности*** $\{a_1, a_2, \ldots, a_n\}$, если для любого сколь угодно малого ε можно указать такое достаточно большое положительное число N, что для всех $n > N$ выполняется неравенство $|a_n - x| < \varepsilon$.

Многие из математических величин или значений функций могут быть выражены как сумма таких бесконечных последовательностей. Например, функции $\sin(x)$ и $\cos(x)$:

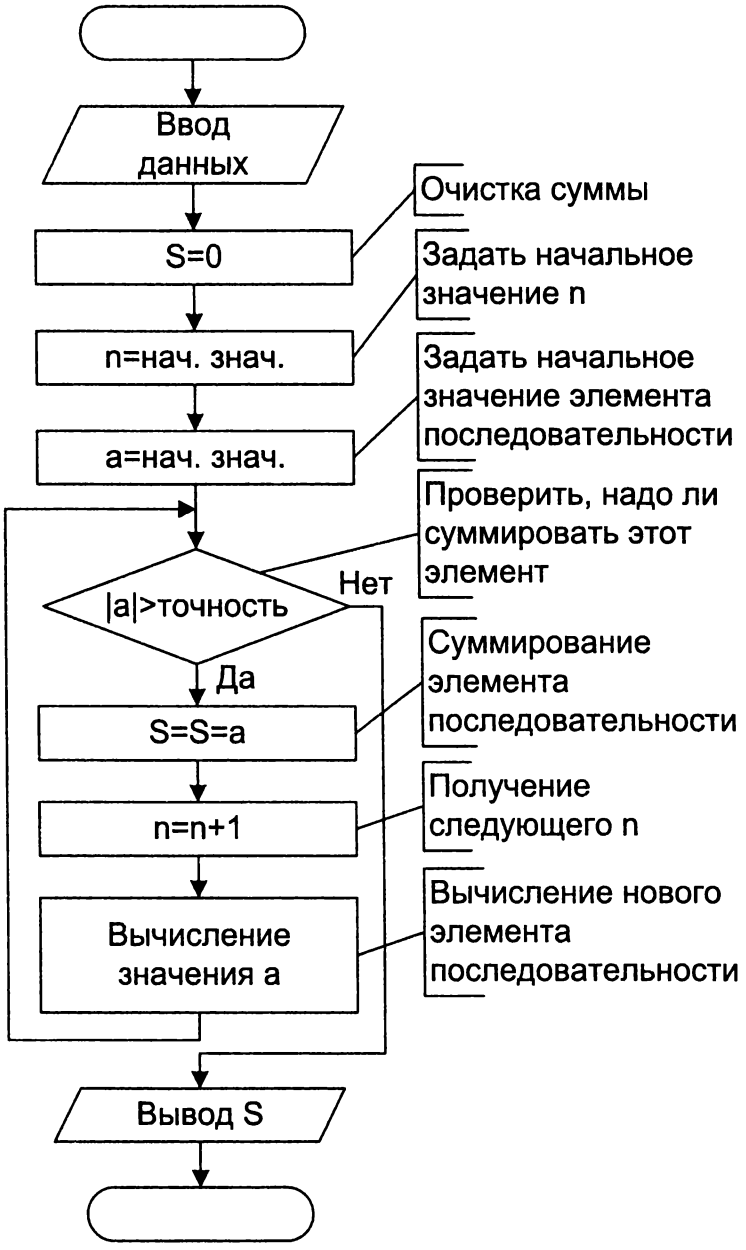
$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!} + \ldots,$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots (-1)^n \cdot \frac{x^{2n}}{(2n)!} + \ldots .$$

Чем больше членов ряда участвуют в вычислении суммы, тем более точным получается результат. Разность между суммой ряда и суммой бесконечного ряда называется ***погрешностью сложения***. Часто оценивают n-й член; если он достаточно мал, т. е. меньше некоторого числа ε (его часто называют точностью), считается, что найденная сумма достаточно при-

ближена к действительному значению суммы и следующие слагаемые можно не учитывать.

Блок-схема алгоритма вычисления суммы с заданной точностью ε выглядит следующим образом:



Примеры

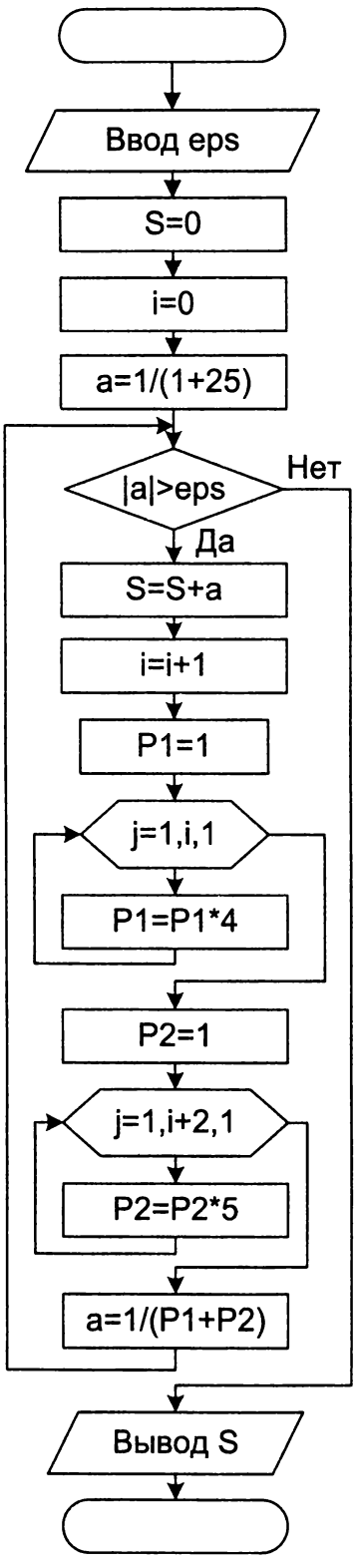
Пример 1

Вычислить бесконечную сумму с заданной точностью ε (ε > 0): $\sum_{i=0}^{\infty} \frac{1}{4^i + 5^{i+2}}$.

Исходные данные: точность eps – вещественный тип, *a* – член последовательности – вещественный тип.

Результат: S – сумма – вещественный тип.

Тестовый пример: при eps = 10 – 4 S = 0,0097.




```
program sum;
 var eps, a, S, P1, P2:real
     i, j integer;
begin
   write(' eps=');
   readln(eps);
   S;=0;

   i:=0;

   a:=1/(1+25);
      while abs(a)>eps do
          begin


   S:=S+a;

  i:=i+1;

  P1:=1;

    for j:=1 to I do

  P1:=P1*4;


  P2:=1;

    for j:=1 to i+2 do

  P2:=P2*5;


  a:=1/(P1+P2);

  end;
    writeln('S=', S:7:3);

end.
```

При вычислении суммы бесконечного ряда члены ряда с увеличением номера стремятся к нулю. Это происходит потому, что значение знаменателя быстро растет и в конце концов достигает очень большого значения, что при-

водит к ошибке переполнения. Чем выше точность, тем легче получить такую ошибку. В некоторых ситуациях этого можно избежать, если использовать рекуррентную формулу, т. е. если выразить новый член ряда через предыдущий.

Рассмотрим примеры построения рекуррентной формулы.

Пример 2

Даны действительные числа x, ε ($x \neq 0$, $\varepsilon > 0$). Вычислить с точностью ε: $\sum_{k=0}^{\infty} \frac{x^{2k}}{2^k k!}$.

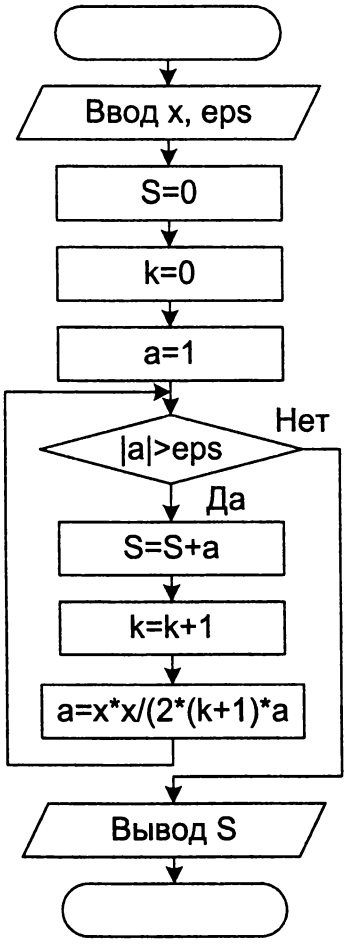
Попробуем выразить формулу для a_{k+1} через a_k: $a_k = \frac{x^{2k}}{2^k k!}$, $a_{k+1} = \frac{x^{2(k+1)}}{2^{k+1}(k+1)!} = \frac{x^{2k} \cdot x^2}{2^k \cdot 2 \cdot k!(k+1)} = \frac{x^{2k}}{2^k k!} \cdot \frac{x^2}{2(k+1)} = a_k \frac{x^2}{2(k+1)}$.

Таким образом, $a_{k+1} = \frac{x^2}{2(k+1)} a_k$.

Исходные данные: x – вещественный тип, eps – вещественный тип, a – член последовательности – вещественный тип.

Результат: S – сумма – вещественный тип.

Тестовый пример: при eps = 10 – 4, $x = 1$ $S = 1{,}8244$.



Рассмотрим еще один пример построения рекуррентной формулы:

$$a_k = \frac{(-1)^k x^{4k+3}}{(2k+1)!\cdot(4k+3)},$$

$$a_{k+1} = \frac{(-1)^{k+1} x^{4(k+1)+3}}{(2(k+1)+1)!\cdot(4(k+1)+3)} = \frac{(-1)^k\cdot(-1)\cdot x^{4k+4+3}}{(2k+3)!\cdot(4k+7)} =$$

$$= \frac{(-1)^k\cdot(-1)\cdot x^{4k+3}\cdot x^4}{(2k+1)!\cdot(2k+2)\cdot(2k+3)\cdot(4k+7)} =$$

$$= \frac{(-1)^k\cdot x^{4k+3}}{(2k+1)!}\cdot\frac{(4k+3)}{(4k+3)}\cdot\frac{(-1)\cdot x^4}{(2k+2)\cdot(2k+3)\cdot(4k+7)} =$$

$$= a_k\cdot\frac{(-1)\cdot x^4\cdot(4k+3)}{(2k+2)(2k+3)(4k+7)}.$$

В некоторых числовых последовательностях требуется получать элементы до тех пор, пока разность между ними не достигнет заданной точности $|a_n - a_{n-1}| < \varepsilon$. В этом случае следует сохранять в памяти два элемента последовательности.

Пример 3

Дано действительное число ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону:

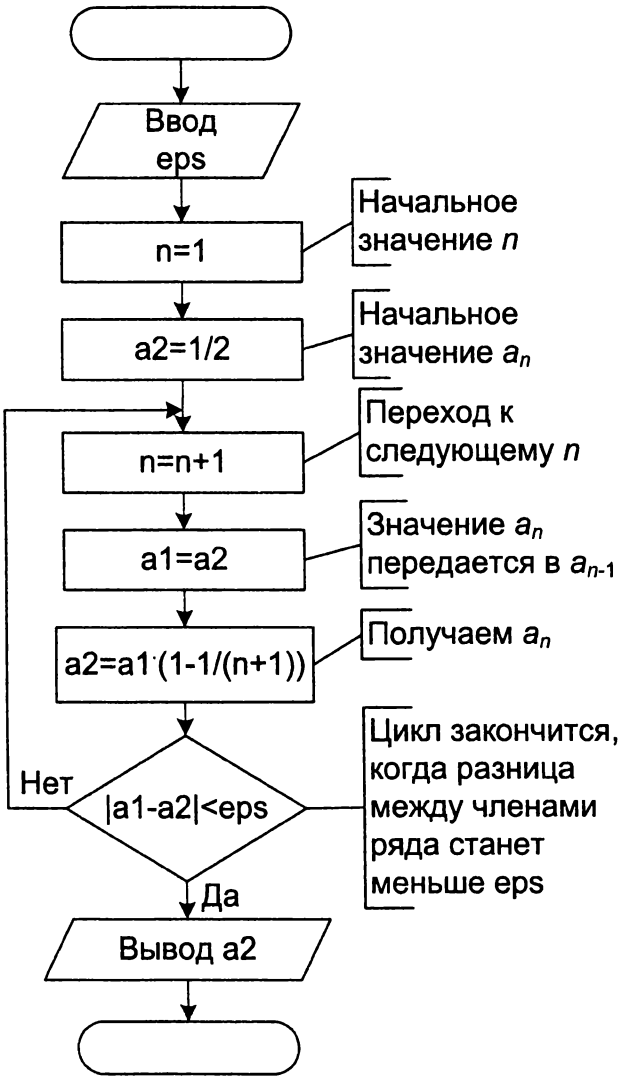
$$a_n = \left(1-\frac{1}{2}\right)\cdot\left(1-\frac{1}{3}\right)\cdot\ldots\cdot\left(1-\frac{1}{n+1}\right).$$

Найти первый член a_n ($n \geqslant 2$), для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$.

Исходные данные: *eps* – вещественный тип, a_{n-1} a1 – элемент – вещественный тип, a_n a2 – элемент – вещественный тип.

Результат: *a*2 – элемент – вещественный тип.

Тестовый пример: при eps = 10 – 4 *a*2 = 0,14.



Контрольные вопросы

1. Почему при вычислении суммы бесконечной последовательности количество членов можно ограничить?

2. Почему в проверке на достижение точности член ряда указан по модулю?

3. Что произойдет, если член ряда с увеличением *n* будет возрастать?

4. Почему рекомендуется получать рекуррентное соотношение для вычисления суммы ряда, содержащего факториал и возведение в целую степень?

5. Можно ли в примере 1 построить рекуррентное соотношение?

6. Если в примере 2 условием выхода из цикла будет не значение элемента меньше ε, а значение разности меньше ε, то количество слагаемых будет больше или меньше?

Задания для лабораторной работы

1. Даны действительные числа x, ε ($x \neq 0$, $\varepsilon > 0$). Вычислить с точностью ε

$$\sum_{k=0}^{\infty} \frac{(-1)^{k+1} x^{2k-1}}{(2k-1)(2k+1)!}.$$

2. Даны целое число n, действительные числа x, ε ($x \neq 0$, $\varepsilon > 0$). Вычислить с точностью ε

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{k!(k+n)!}\left(\frac{x}{2}\right)^{n+2k}.$$

3. Дано действительное число x. Вычислить с точностью 10^{-6}

$$\sum_{k=1}^{\infty} \frac{(-1)^k x^k}{k}.$$

4. Даны действительные числа x, ε ($x \neq 0$, $\varepsilon > 0$). Вычислить с точностью ε бесконечную сумму $\sum_{k=0}^{\infty} \frac{(-x)^{2k}}{2k!}$ и указать количество учтенных слагаемых.

5. Дано действительное число x ($x<1$). Последовательность a_1, a_2, ... образована по следующему закону:

$$a_n = \frac{x}{\sqrt{n}(n+2)!}.$$

Получить $a_1 + \ldots + a_k$, где k – наименьшее целое число, удовлетворяющее двум условиям: $k > 10$ и $|a_{k+1}| < 10^{-5}$.

6. Даны действительные числа x и ε ($x \neq 0$, $\varepsilon > 0$). Вычислить $\sum_{k=0}^{\infty} \frac{(-1)^k (k+1) x^k}{3^k}$ с точностью ε и указать количество учтенных слагаемых.

7. Дано действительное число ε ($\varepsilon > 0$). Вычислить $\sum_{n=1}^{\infty} \frac{1}{3^n} \cos^3(3^{n-1})$, учитывая только те слагаемые, в которых множитель $1/3^n$ имеет величину не меньшую, чем ε.

8. Дано действительное число ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону:

$$a_n = \frac{n}{\sqrt{n^2+1}+\sqrt{n^2-1}}.$$

Найти сумму a_n до первого члена a_n ($n \geqslant 2$), для которых выполняется условие $|a_n - a_{n-1}| < \varepsilon$.

9. Дано действительное число ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону:

$$a_n = \underbrace{\sqrt{\frac{1}{2}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\cdot\sqrt{\frac{1}{2}}}\cdots\sqrt{\frac{1}{2}+\frac{1}{2}\cdot\sqrt{\frac{1}{2}+\ldots+\frac{1}{2}\cdot\sqrt{\frac{1}{2}}}}}}_{n}.$$

Найти первый член a_n ($n \geqslant 2$), для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$.

10. Дано действительное число ε (ε >0). Последовательность a_1, a_2, … образована по следующему закону:

$$a_n = \left(1-\frac{1}{2!}\right)\left(1+\frac{1}{3!}\right)\ldots\left(1+\frac{(-1)^n}{(n+1)!}\right).$$

Найти первый член a_n ($n \geqslant 2$), для которого выполняется условие $|a_n - a_{n-1}| < \varepsilon$.

11. Даны целое число n, действительные числа x, ε ($x \neq 0$, ε > 0). Последовательность a_1, a_2, … образована по следующему закону: $a_1 = x$; далее для $n = 2, 3, \ldots$ выполнено

$$a_n = \frac{1}{2}\left(a_{n-1}+\frac{x}{a_{n-1}}\right).$$

Вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

12. Даны действительные числа x и ε ($\varepsilon > 0$). Последовательность a_1, a_2, … образована по следующему закону: $a_1 = x$; далее для $n = 2, 3, \ldots$ выполнено

$$a_n = 3 + \frac{1}{2^n}\cos^2(a_{n-1} - x).$$

Вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

Тема 4. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ

Процесс решения сложной задачи часто сводится к решению нескольких более простых подзадач. Соответственно, процесс разработки сложного алгоритма разбивается на этапы составления отдельных алгоритмов, которые называются ***вспомогательными***. Каждый вспомогательный алгоритм описывает решение какой-либо подзадачи.

При разработке сложных алгоритмов используется метод последовательной детализации, который состоит в следующем. Сначала алгоритм формулируется в крупных блоках – их еще нельзя перевести на язык операторов и они записываются как вызовы вспомогательных алгоритмов. Затем происходит детализация, и все вспомогательные алгоритмы подробно расписываются для перевода на язык команд. Вспомогательные алгоритмы на языке программирования реализуются через подпрограммы.

Программы, которые неразделимы на отдельные структурные элементы, называются ***монолитными***. Большие монолитные программы сложны для разработки, отладки и сопровождения. Минимальным автономным элементом монолитной программы является оператор.

Естественно стремление разбить программу на более крупные, чем оператор, компоненты. Роль таких компонентов выполняют подпрограммы. Программы, содержащие подпрограммы, называются ***модульными***. Подпрограммы, в свою очередь, могут вызывать подпрограммы более низкого уровня и т. д. Таким образом, каждая модульная программа имеет иерархическую структуру.

Использование аппарата подпрограмм позволяет сократить объем и улучшить структуру программы с точки зрения наглядности и читаемости, уменьшить вероятность ошибок и облегчить процесс отладки программы. Разложение программы на взаимосвязанные, но замкнутые и логически завершенные компоненты дает возможность выполнять разработку отдельных подпрограмм разным программистам и более или менее независимо друг от друга. Кроме того, подпрограмма может быть рассмотрена как отдельная единица (со своими входными и выходными данными), что позволяет использовать ее в общем иерархическом подходе

при конструировании алгоритма и программы по принципам нисходящего проектирования.

В языке *Pascal* подпрограммы реализуются в виде процедур и функций. Существуют встроенные (стандартные) процедуры и функции, которые являются частью языка и могут вызываться по имени без предварительного описания. Процедуры и функции пользователя пишутся самим программистом в разделе описания процедур и функций. Их вызов для последующего выполнения обычно записывается в разделе операторов. В процедуре или функции возможно вызывать другую процедуру или функцию. При этом вызываемая подпрограмма должна быть описана раньше, чем вызывающая.

Лабораторная работа 7. Функции

Теория

Функция – это подпрограмма, вычисляющая и возвращающая некоторое значение.

Функция пользователя применяется только тогда, когда требуется вычислить единственное значение. Она возвращает результат в точку своего вызова. Поэтому функция является частью какого-нибудь выражения.

Функция, определенная пользователем, описывается в разделе описания процедур и функций после описания переменных. Функция состоит из заголовка и тела. Заголовок функции имеет следущий вид: Function *ИмяФункции* (*ФормальныеПараметры*): *ТипРезультата*; В круглых скобках указывается необязательный список параметров. Имя функции должно быть уникальным в пределах программы.

Тело функции по структуре аналогично обычной программе, т. е. состоит из раздела описаний и раздела операторов. Только, в отличие от раздела операторов программы, после слова *end* ставится точка с запятой. Кроме того, в разделе операторов обязательно должен находиться по крайней мере один оператор, который присваивает имени функции значение, возвращаемое как результат работы функции. Если таких присваиваний несколько, то результатом будет являться последний оператор присваивания.

Данные, описанные в подпрограмме, действительны только в пределах данной пользовательской подпрограммы. Они называются *локальными*. Данные, описанные в основной программе, называются *глобальными*, и их также можно использовать в подпрограмме. Если имя глобальной переменной совпадает с именем локальной, внутри подпрограммы эта переменная интерпретируется как локальная.

Параметры функции позволяют вычислять функцию с различными начальными данными. Параметры указываются в заголовке функции и называются *формальными* параметрами. Описание формальных параметров имеет следующий вид: *имя параметра: тип*. Описания параметров разных типов разделяются точкой с запятой. Каждый формальный параметр является локальным по отношению к подпрограмме.

При вызове функции формальные параметры заменяются фактическими. Формальные и фактические параметры должны соответствовать друг другу:

- по количеству (количество формальных и фактических параметров должно быть одинаково);
- по последовательности (первому формальному параметру должен соответствовать первый фактический параметр, второму – второй и т. д.);
- по типу (типы соответствующих формальных и фактических параметров должны совпадать).

Фактический параметр-аргумент может быть выражением соответствующего типа. Для досрочного выхода из подпрограммы используется оператор *exit*.

Графически вспомогательный алгоритм разрабатывается как отдельная блок-схема, но в блоке «Начало» указывается заголовок подпрограммы.

Примеры

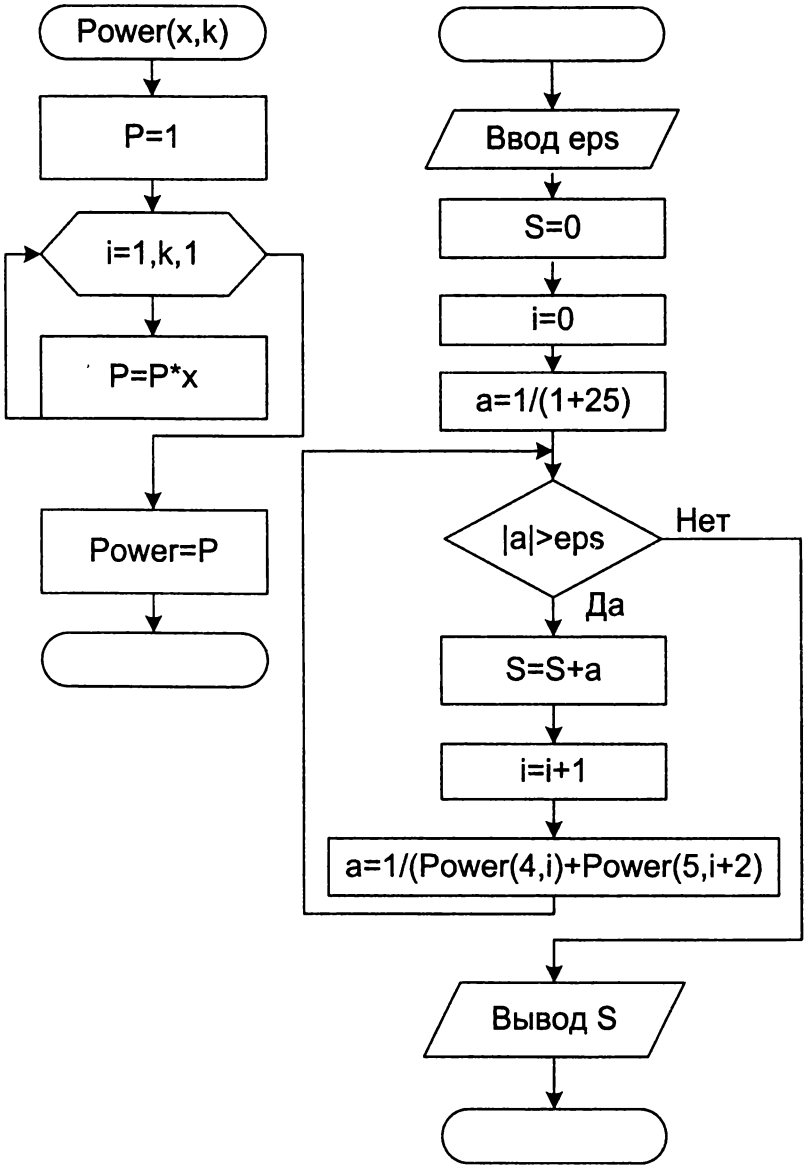
Пример 1

Вычислить $\sum_{i=0}^{\infty} \frac{1}{4^i + 5^{i+2}}$ с заданной точностью ε.

Исходные данные: eps – точность – вещественный тип, a – член последовательности – вещественный тип.

Результат: S – сумма – вещественный тип.

Тестовый пример: при eps = 10^{-4} S = 0,0097.




```
program sum;
 var eps, a, S:real
     i: integer;
  Function Power(x: real;
k: integer):real;
              var P: real;
                   i:integer;
    begin
      P:=1;
      For i:=1 to k do
           P:=P*x;
      Power:=P;
    end;
begin
  write(' eps=');
  readln(eps);
   S;=0;
    i:=0;
    a:=1/(1+25);
    while abs(a)>eps do
         begin
              S:=S+a;
               i:=i+1;
                a:=1/
(Power(4,i)+Power(5,i+2));
      end;
      writeln('S=', S:7:3);
end.
```

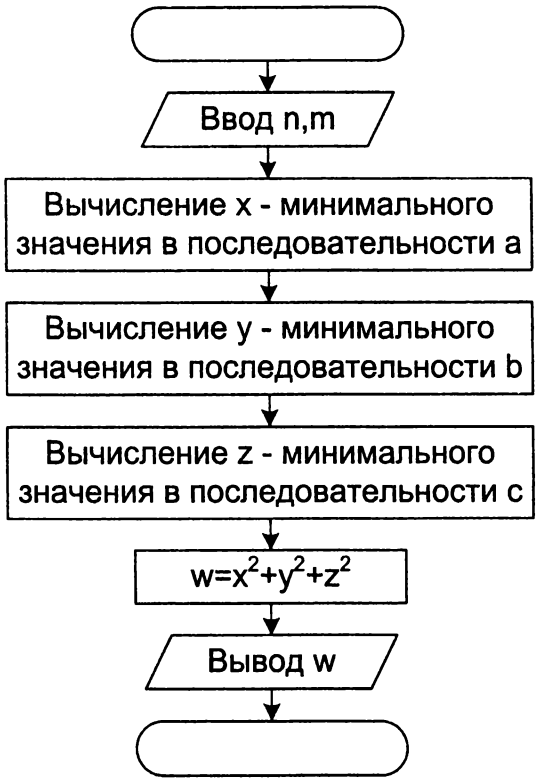
Пример 2

Даны натуральные числа n, m, целые числа $a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_{10}$. Получить $w = x^2 + y^2 + z^2$, где $x = \min(a_1, \ldots, a_n)$, $y = \min(b_1, \ldots, b_m)$, $z = min(c_1, \ldots, c_{10})$.

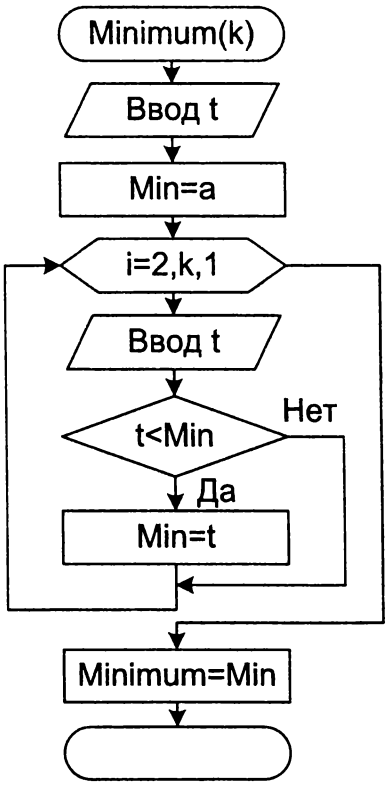
Исходные данные: n, m – целый тип, a – элемент последовательности – целый тип, b – элемент последовательности – целый тип, c – элемент последовательности – целый тип.

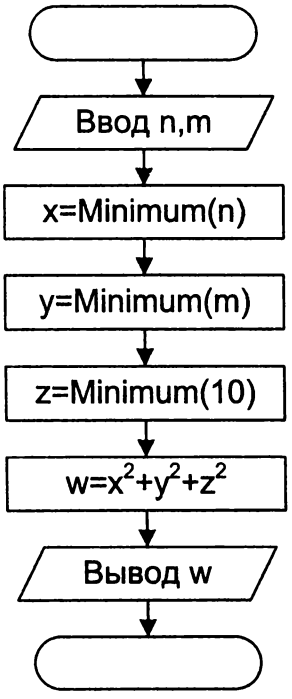
Результат: w – целый тип.

Блок-схема алгоритма функции нахождения минимального значения выглядит следующим образом:



Тестовый пример: при $n = 7$, $m = 8$ последовательность a имеет следующий вид: 5, 2, 7, 0, 6, 1, 4; последовательность b: –1, –3, –5, –3, –6, –2, –2, –7; последовательность c: –2, 4, 6, –8, 3, 5, –3, –5, –2, 1; $w = 65$.





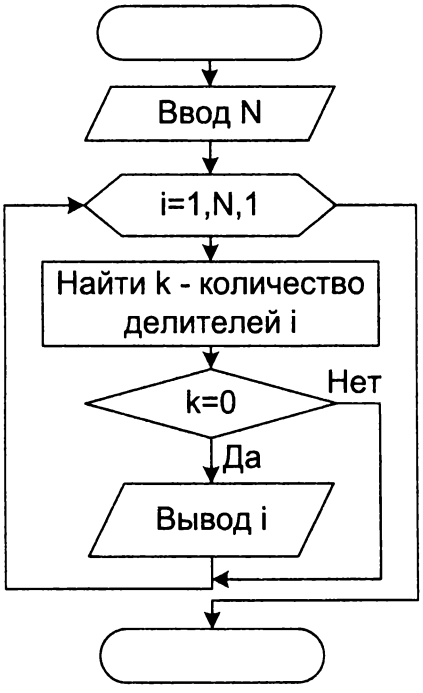
Пример 3

Составить программу нахождения всех простых чисел, не превосходящих *N*.

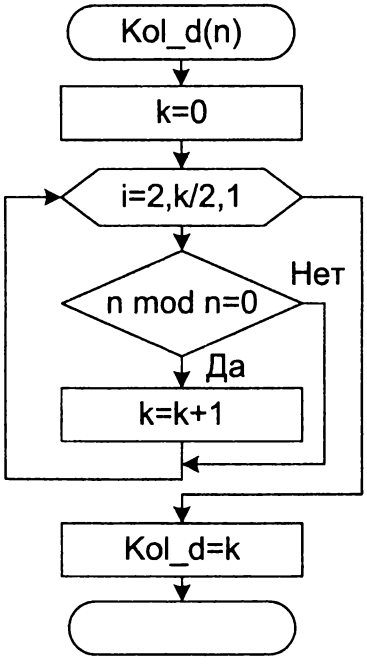
Исходные данные: *N* – целый тип.

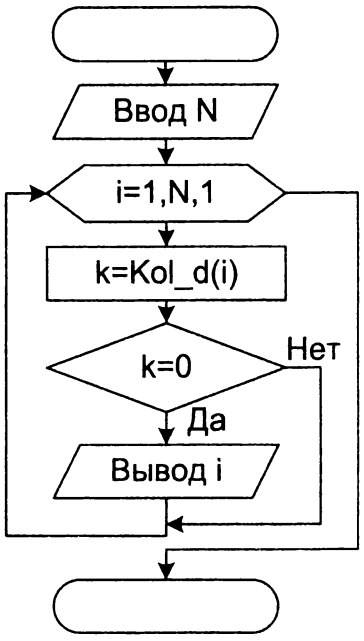
Результат: вывод *i* чисел, количество делителей у которых равно 0.

При построении укрупненной блок-схемы не выявлены повторяющиеся задачи, но подсчет количества делителей у числа следует рассматривать как самостоятельную задачу, для которой можно написать отдельную функцию:



Тестовый пример: при *N* = 15 вывод «1, 2, 3, 5, 7, 11, 13».





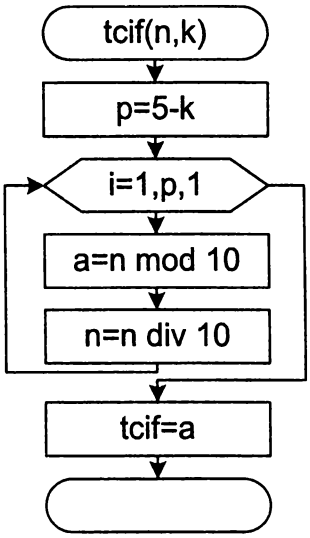
Пример 4

Написать программу, которая проверяет, являются ли во введенном четырехзначном числе вида $\overline{a}\,\overline{b}\,\overline{c}\,\overline{d}$ все цифры разными.

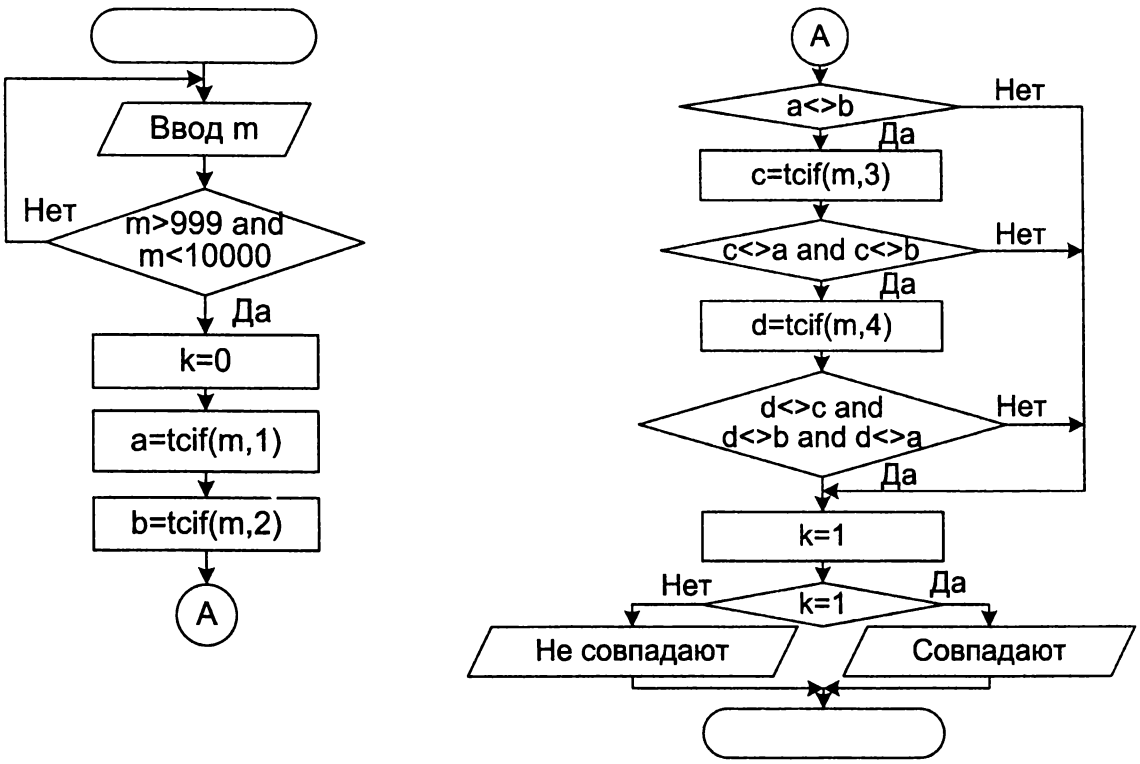
Исходные данные: четырехзначное число m.

Результаты: a, b, c, d – цифры числа m – целый тип; $k = 0$, если есть совпадение цифр; $k = 1$, если совпадения цифр нет.

Следует создать функцию, которая выделяет заданную цифру из обозначенного числа. Эта функция работает только с 4-значным числом, поэтому в начале программы следует проверить количество цифр в числе.



Тестовый пример: при 2467 – все цифры разные; при 1233 – цифры совпадают.



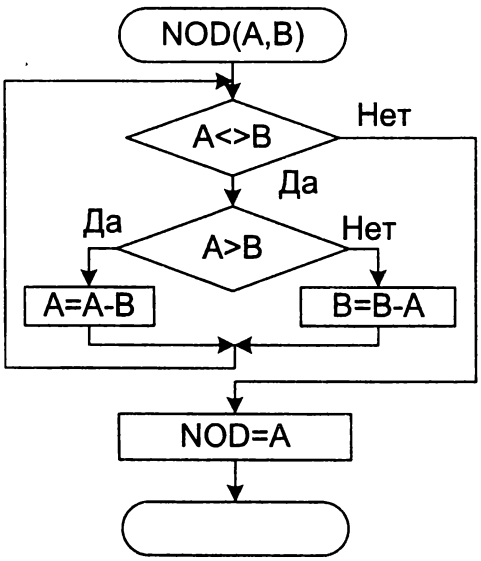
Пример 5

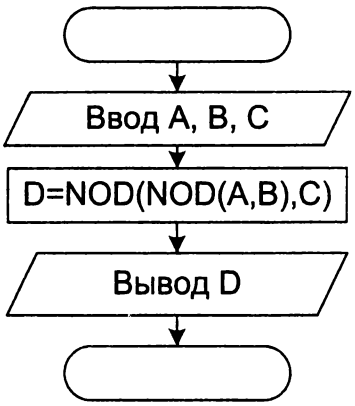
Найти наибольший общий делитель трех чисел.

Исходные данные: *A*, *B*, *C* – целый тип.

Результат: *D* – наибольший общий делитель чисел *A*, *B*, *C* – целый тип.

Нахождение наибольшего общего делителя (НОД) двух чисел выполняется в функции NOD. Сначала находится НОД для чисел *A*, *B*, а затем для полученного значения и *C*.





Контрольные вопросы

1. Может ли функция не иметь параметров?

2. Могут ли в качестве фактических параметров использоваться выражения?

3. Какие правила соответствия должны выполняться для формальных и фактических параметров?

4. Что происходит, если имя формального параметра совпадает с именем глобального?

5. В примере 3 цикл с параметром *i* используется и в основной программе, и в подпрограмме. Не запутается ли программа, столкнувшись с таким совпадением?

6. Можно ли в теле функции использовать несколько операторов присваивания имени функции выражения?

7. Можно ли из одной функции вызывать другую?

Задания для лабораторной работы

1. Даны действительные числа s, t. Получить

$$h(s, t) + \max(h^2(s - t, st), h^4(s - t, s + t)) + h\,(1,1),$$

где $h(a, b) = \dfrac{a}{1 + b^2} + \dfrac{b}{1 + a^2} - (a - b)^3$.

2. Даны действительные числа a, b, натуральное число n ($b > a$). Получить $(f_1 + \ldots + f_n)h$, где $h = \dfrac{b-a}{n}$, $f_i = \dfrac{a + \left(i - \dfrac{1}{2}\right)h}{1 + \left(a + \left(i - \dfrac{1}{2}\right)h\right)^2}$, $i = 1, 2, \ldots, n$.

3. Даны натуральные числа k, l, m, целые числа $x_1, \ldots, x_k, y_1, \ldots, y_l, z_1, \ldots, z_m$. Получить

$$l = \begin{cases} (\max(y_1, \ldots, y_l) + \max(z_1, \ldots, z_m))/2 & \text{при } \max(x_1, \ldots, x_k) >= 0, \\ 1 + (\max(x_1, \ldots, x_k))^3 & \text{в противном случае.} \end{cases}$$

4. Дано натуральное число n. Вычислить $\sum\limits_{k=1}^{n} \dfrac{k!}{S_k}$, где $S_k = \dfrac{1}{2} + \dfrac{1}{3} + \cdots + \dfrac{1}{k+1}$.

5. Даны натуральные числа u, v и n. Найти $\sum\limits_{k=1}^{n} \dfrac{a_k b_k}{(k+1)!}$, где $a_1 = u$, $b_1 = v$, $a_k = 2b_{k-1} + a_{k-1}$; $b_k = 2a^2_{k-1} + b_{k-1}$, $k = 2, 3, \ldots$. Вычисление a_k и b_k и факториала выполнять с помощью функции.

6. Даны действительные числа x и ε ($x \neq 0$, $\varepsilon > 0$). Вычислить $\sum\limits_{k=0}^{\infty} \dfrac{(-1)^{k+1} x^{2k-1}}{(2k-1)!(2k+1)!}$ с точностью ε и указать количество учтенных слагаемых.

Возведение в степень и факториал вычислять с помощью функций.

7. Дано действительное число y. Получить $\dfrac{1{,}7t(0{,}25) + 2t(1+y)}{6 - t(y^2 - 1)}$, где

$$t(x) = \frac{\sum\limits_{k=0}^{10} \dfrac{x^{2k+1}}{(2k+1)!}}{\sum\limits_{k=0}^{10} \dfrac{x^{2k}}{(2k)!}}.$$

8. Составить программу для нахождения чисел из интервала [*M*, *N*], имеющих наибольшее количество делителей.

9. Составить программу, определяющую, в каком из двух данных чисел больше цифр.

10. Два натуральных числа называются дружественными, если каждое из них равно сумме всех делителей (кроме его самого) другого числа (например, 220 и 284). Найти все пары дружественных чисел, которые не больше данного числа *N*.

11. Натуральное число, в записи которого *n* цифр, называется числом Амстронга, если сумма его цифр, возведенная в степень *n*, равна самому числу. Найти все числа Амстронга от 1 до *k*.

12. Найти все натуральные числа, не превосходящие *n*, которые делятся на каждую из своих цифр.

Лабораторная работа 8. Процедуры

Теория

Процедура – это независимая именованная часть программы, которую после однократного описания можно многократно вызывать по имени из последующих частей программы для выполнения определенных действий.

Процедуры пользователя описываются в разделе описания процедур и функций перед операторами программы. Структура процедуры повторяет структуру программы и состоит из заголовка процедуры, раздела описания локальных переменных и операторов, реализующих алгоритм процедуры. Заголовок процедуры имеет следующий вид: procedure *ИмяПроцедуры (ФормальныеПараметры)*

В конце процедуры, после *end*, ставится точка с запятой.

Для обращения к процедуре используется оператор вызова процедуры. Он состоит из имени процедуры и списка фактических параметров, которые отделяются друг от друга запятыми и заключаются в круглые скобки.

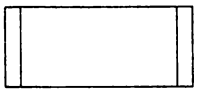
Процедуры имеют более широкий спектр применения, чем функции:

- с помощью процедуры можно одновременно вычислять несколько значений;
- в процедуре можно изменять входные параметры;
- результатом процедуры может быть не только вычисление значений, но и выполнение каких-нибудь действий, например, ввод или вывод данных.

Значения, которые вычисляются в процедуре, должны быть представлены как дополнительные параметры процедуры, но, в отличие от входных параметров, от которых зависит вычисление процедуры, перед вычисляемыми параметрами надо ставить слово «*var*». Это *параметры-переменные*. Все изменения в процедуре параметров-переменных отражаются на фактических параметрах. Поэтому параметрами-переменными в процедуре являются не только вычисляемые значения, но и изменяемые. Остальные входные параметры называются *параметрами-значениями*. Они действуют как локальные переменные, поэтому все изменения в процедуре параметров-значений не влияют на значения соответствующих фактических параметров.

Досрочный выход из процедуры – оператор *exit*.

В блок-схеме программы блок-схема процедуры, как и блок-схема функции, изображается отдельно. В блок-схеме основной программы оператор вызова подпрограммы изображается в виде блока:



Примеры

Пример 1

Дана дробь $\frac{A}{B}$ (A, B – натуральные числа). Представить эту дробь в виде несократимой дроби.

Исходные данные: A – числитель – целый тип, B – знаменатель – целый тип.

Результат: A – числитель после сокращения, B – знаменатель после сокращения.

Чтобы сократить дробь, нужно найти НОД для числителя и знаменателя, а затем разделить их на найденное число.

Сокращение дроби запишем в отдельной процедуре SOKR. Эта процедура в дальнейшем может быть использована в других задачах.

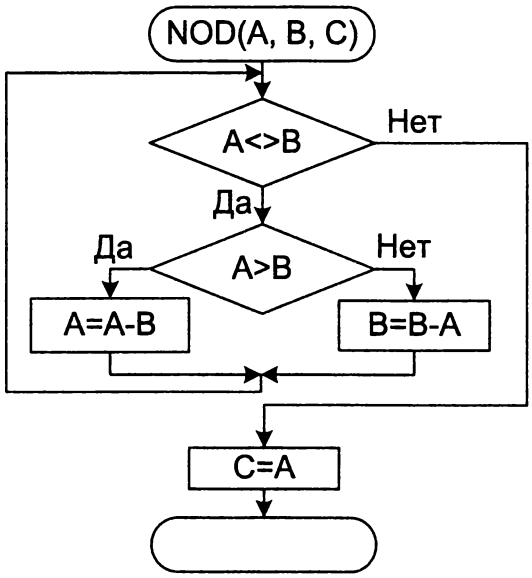
Данную задачу можно решить только с использованием процедуры, так как в результате этой подпрограммы изменяются входные параметры.

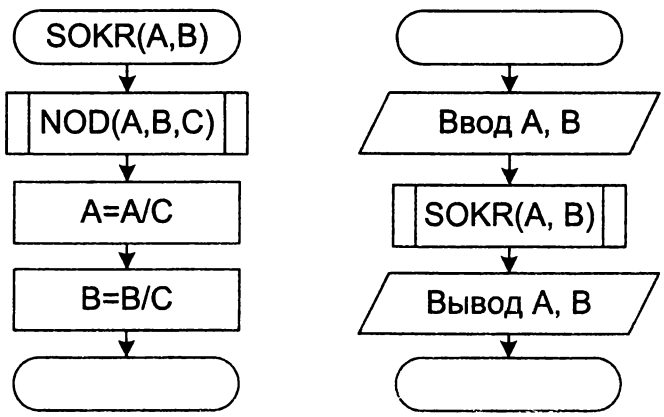
Обратите внимание на то, что функция NOD, которая была рассмотрена в предыдущей лабораторной работе, в данном примере представлена как процедура.

Любая функция может быть представлена как процедура. В этом случае полученный результат является еще одним параметром процедуры, причем параметром-переменной.

В данном примере используются две процедуры: процедура NOD для нахождения наибольшего общего делителя и процедура SOKR для сокращения числителя и знаменателя. Процедура SOKR вызывает процедуру NOD, поэтому в разделе описания процедур процедура NOD должна предшествовать процедуре SOKR.

Тестовый пример: при $A = 7$, $B = 56$ после сокращения получаем $A = 1$, $B = 8$.






```
program proc;
var A, B: integer;
{Процедура нахождения наибольшего общего делителя}
A и B, наибольший общий делитель – C
procedure NOD (A, B: integer; var C: integer);
```

```
begin
while A<>B do
if A>B then A:=A-B
else B:=B-A;
C:=A;
end;
{Процедура сокращения числителя A и знаменателя B}
procedure SOKR (var A, B: integer);
var C: integer;
begin
NOD (A, B, C);
A:=A/C;
B:=B/C:
end;
{Основная программа}
begin
write ('Введите числитель и знаменатель')
readln (A, B);
SOKR (A, B);
writeln ('Числитель =', A, 'Знаменатель =', B);
readln;
end.
```

Пример 2

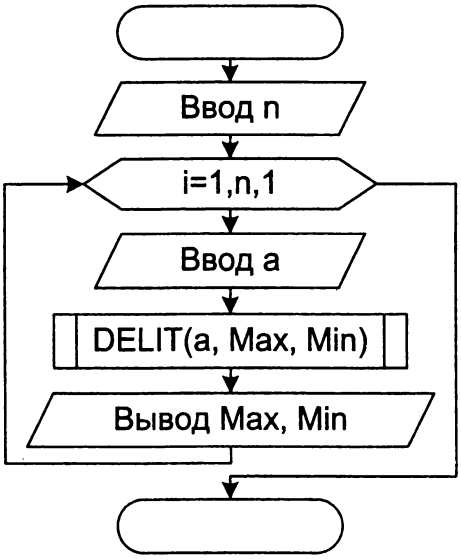
Дана последовательность из n натуральных чисел. Для каждого числа указать его наименьший и наибольший делитель. Для простого числа – 1 и само число, для остальных это должны быть числа, отличные от 1 и самого числа.

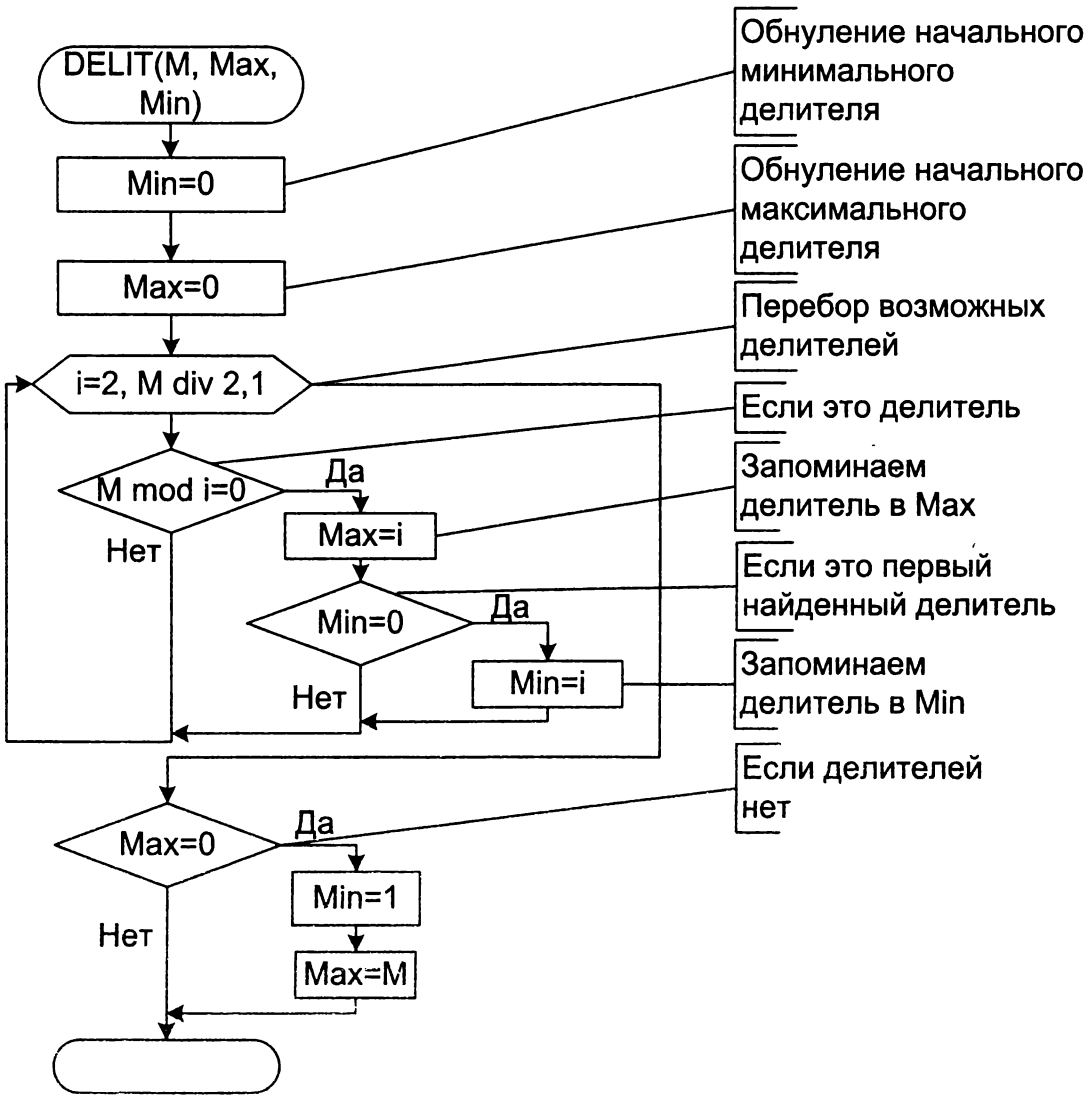
Исходные данные: n – количество элементов последовательности, a – элемент последовательности.

Результат: *Max* – наибольший делитель, *Min* – наименьший делитель каждого элемента последовательности.

Нахождение наибольшего и наименьшего делителя выполняется в процедуре DELIT. Процедура в этом случае используется потому, что в результате получается не одно значение, а два.

Тестовый пример: при $n = 5$ и $a = 6$ $Max = 3$, $Min = 2$; $a = 7$ $Max = 7$, $Min = 1$; $a = 24$ $Max = 12$, $Min = 2$; $a = 15$ $Max = 5$, $Min = 3$; $a = 63$ Max = 7, Min = 3.


Ввод n
i=1,n,1
Ввод a
DELIT(a, Max, Min)
Вывод Max, Min


DELIT(M, Max, Min)
Min=0
Max=0
i=2, M div 2,1
M mod i=0
Да
Нет
Max=i
Min=0
Да
Нет
Min=i
Max=0
Да
Нет
Min=1
Max=M
Обнуление начального минимального делителя
Обнуление начального максимального делителя
Перебор возможных делителей
Если это делитель
Запоминаем делитель в Max
Если это первый найденный делитель
Запоминаем делитель в Min
Если делителей нет

```
program prim2;
var n, i, a, Max, Min. integer;
procedure DELIT (M: integer; var Max, Min: integer);
var i: integer;
begin
Max:=0;
Min:=0;
for i:=2 to M div 2 do
if M mod i=0 then
begin
Max:=I;
if Min=0 then Min:=I;
end;
if Max=0 then
begin Min:=1; Max:=M; end;
end;
begin
write ('n='); readln (n);
for i:=1 to n do
begin
write ('Введите элемент последовательности');
readln (a);
DELIT (a, Max, Min);
writeln ('Max= ', Max, 'Min= ', Min);
end;
end.
```

Пример 3

Найти все натуральные числа, не превосходящие заданное *N*, которые делятся на каждую из своих цифр.

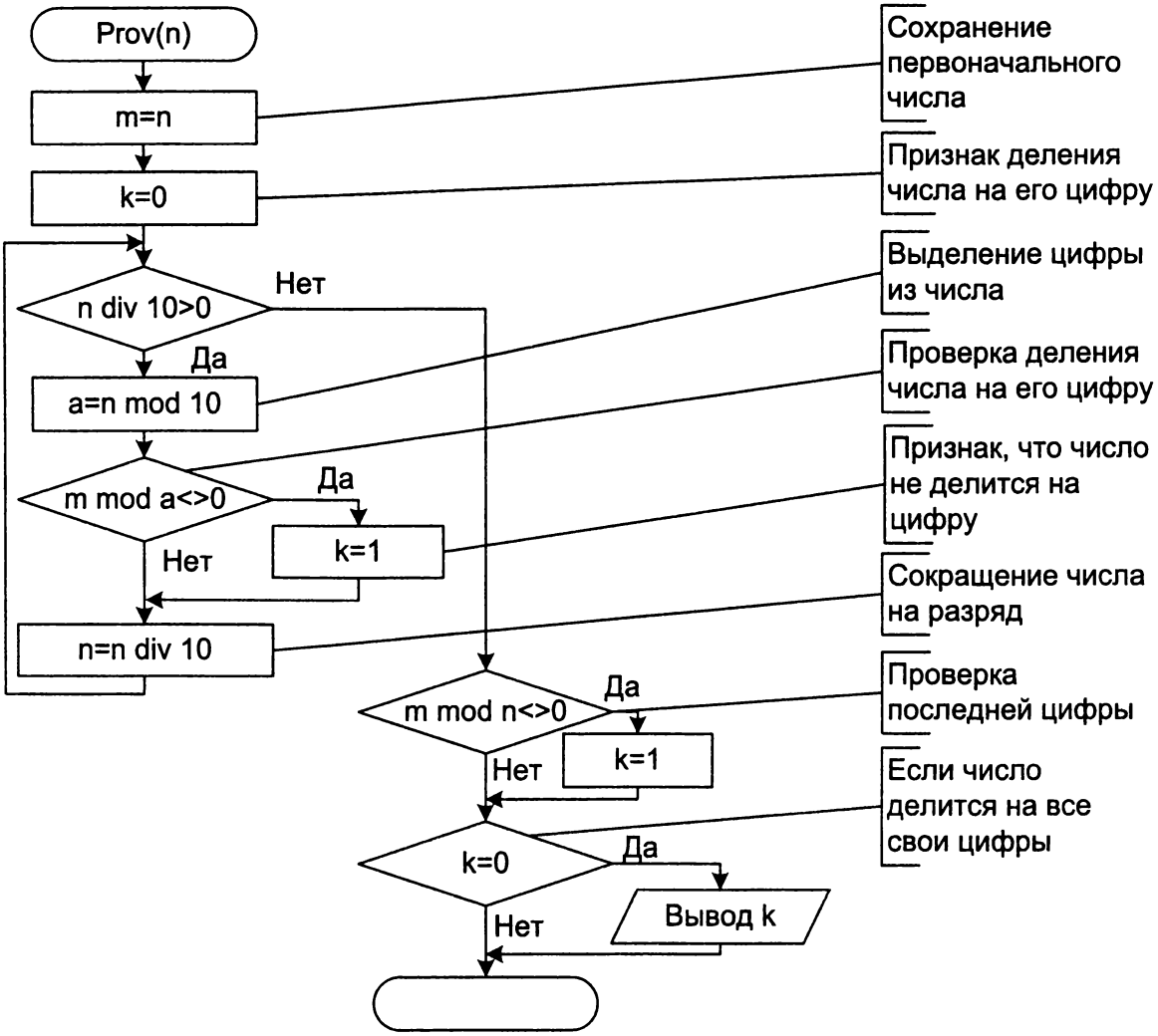
Исходные данные: *N* – целый тип.

Результат: вывод чисел, которые делятся на каждую из своих цифр.

В процедуре *Prov* проверяется, делится ли число на все свои цифры. Если делится – данное число выводится на экран. Результатом процедуры является вывод чисел, удовлетворяющих заданному условию.

Тестовый пример: при $N = 20$ выводятся числа 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20.

Приведен пример блок-схемы процедуры *Prov*, которая проверяет деление числа на свои цифры.



Контрольные вопросы

1. Можно ли в примере 1 при вычислении NOD параметры A и B описать как параметры-переменные?

2. Можно ли в примере 1 поменять местами описания процедур NOD и SOKR?

3. Перечислите преимущества процедур перед функциями.

4. Чем параметры-переменные отличаются от параметров-значений?

5. Можно ли в примере 2 использовать не процедуру, а функцию?

6. В каких случаях процедуру можно заменить на функцию?

7. Можно ли в качестве фактических параметров-переменных использовать выражения?

8. В каких ситуациях вместо процедуры удобнее использовать функцию?

Задания для лабораторной работы

1. Даны дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D – натуральные числа). Составить программу вычитания этих дробей. Результат должен быть несократимой дробью.

2. Даны дроби $\frac{A}{B}$ и $\frac{C}{D}$ (A, B, C, D – натуральные числа). Составить программу для деления этих дробей. Результат должен быть несократимой дробью.

3. Даны натуральные числа n, m и последовательности вещественных чисел $x_1, x_2, \ldots, x_n$; $y_1, y_2, \ldots, y_m$. Найти разности максимумов и минимумов каждой последовательности.

4. Два простых числа называются близнецами, если они отличаются друг от друга на 2 (например, 41 и 43). Напечатать все пары близнецов из отрезка $[n, 2n]$, где n – заданное натуральное число.

5. Дано натуральное n-значное число. Оставить в этом числе только первую цифру, а остальные заменить нулями.

6. Дано натуральное число N. Вывести все совершенные числа, которые меньше или равны N. Совершенным называется число, равное сумме своих делителей.

7. Даны последовательность из n натуральных чисел и натуральное число A. Найти в данной последовательности число, которое имеет самый большой наибольший общий делитель с числом A.

8. Дано натуральное число N. Найти и вывести все числа в интервале от 1 до $N - 1$, у которых сумма всех цифр совпадает с суммой цифр данного числа.

9. Найти все натуральные n-значные числа, цифры в которых образуют строго возрастающую последовательность (например, 1234).

10. Из заданного числа вычли сумму его цифр. Из результата вновь вычли сумму его цифр и т. д. Сколько таких действий надо произвести, чтобы получить нуль?

11. Для последовательности $a_1 = 1$, $a_{n+1} = a_n + \frac{1}{1 - a_n}$ составить программу печати k-го члена в виде обыкновенной несократимой дроби.

12. Дано четное число $n > 2$. Проверить для него гипотезу Гольдбаха: каждое четное n представляется в виде суммы двух простых чисел.

Лабораторная работа 9. Рекурсия

Теория

Рекурсия – это такой способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих ее операторов обращается сама к себе.

В рекурсивной процедуре или функции обязательно должно содержаться условие окончания рекурсии.

При выполнении рекурсии, когда подпрограмма доходит до рекурсивного вызова, процесс приостанавливается и новый процесс запускается с начала, но уже на новом уровне. Прерванный же процесс запоминается. Новый процесс тоже может быть приостановлен и т. д. Образуется последовательность прерванных процессов. Последовательность рекурсивных обращений обязательно должна выходить на определенное значение. При каждом очередном входе в подпрограмму ее локальные параметры и адреса возврата размещаются в специальной области памяти – стеке. После выхода на определенное значение выполняется обратный ход – цепочка вычислений по обратному маршруту, сохраненному в стеке.

Использование рекурсивных подпрограмм – красивый прием с точки зрения программирования. Программа выглядит более компактно. Но у рекурсии есть недостатки: она выполняется более медленно, а ее глубина ограничена.

Примеры

Пример 1

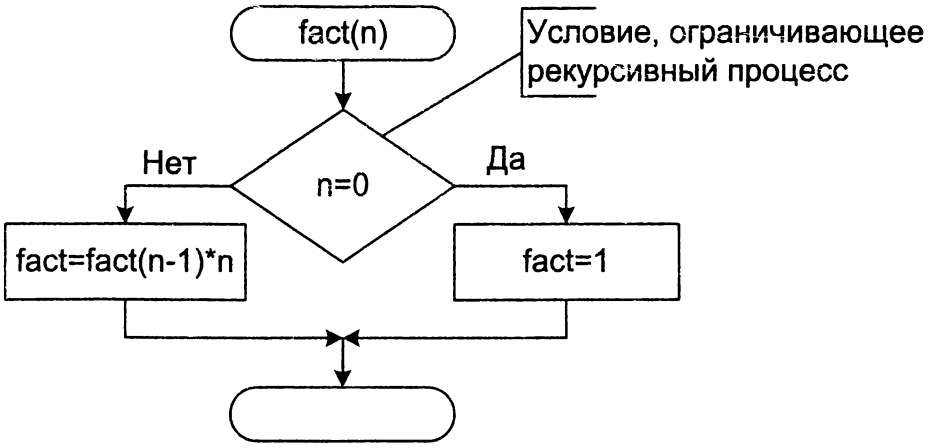
Вычислить факториал *n* с помощью рекурсии.

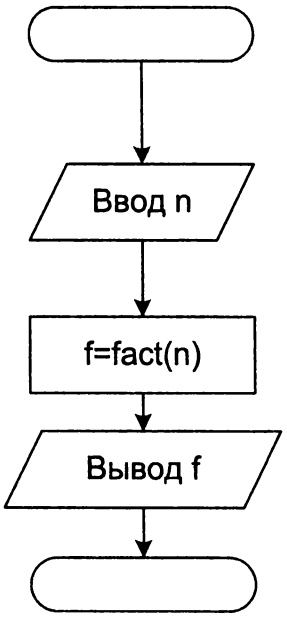
Исходные данные: *n* – целый тип.

Результат: Факториал *f* – вещественный тип. Несмотря на то, что перемножаются данные целого типа, конечный результат очень быстро растет и, если программа написана для *Turbo Pascal*, результат выходит за пределы допустимого диапазона для целого типа *integer*.

Вычисление факториала выполняется через рекурсивную функцию *fact*: *fact(n)=fact(n-1)*n*. Граничные условия рекурсии: *fact*(0)=1.

Тестовый пример: при *n* = 5 *f* = 120.






```
program recurc1;
  var n: integer;
      f: real;
  function fact(n: integer): real;
    begin
      if n=0 then fact:=0
          else fact:=fact(n-1)*n;
    end;
    begin
      write('n='); readln(n);
      f:=fact(n);

 writeln('n!=', n:5:0);


end.
```

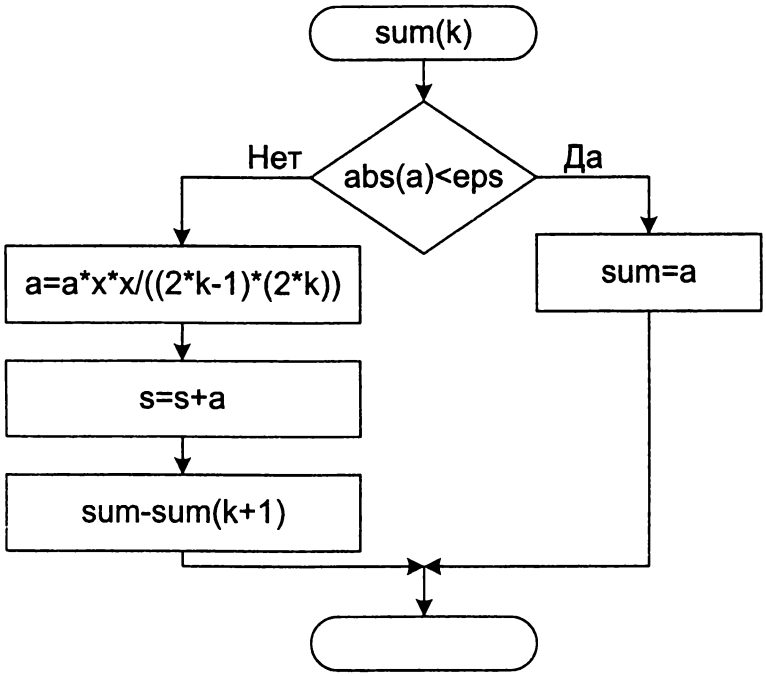
Пример 2

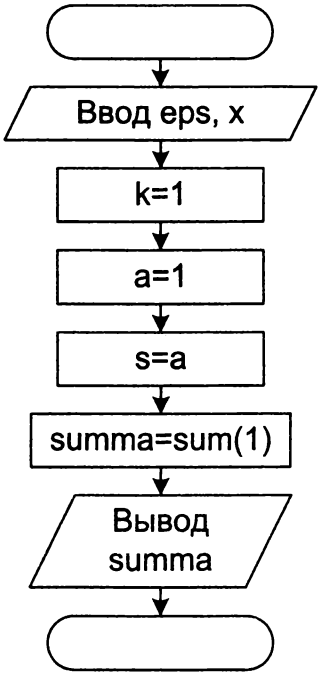
Даны действительные числа x и ε ($x \neq 0$, $\varepsilon > 0$). Вычислить $\sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$ с точностью ε и указать количество учтенных слагаемых. Вычисление выполнить через рекурсию.

Исходные данные: x – вещественный тип, точность eps – вещественный тип.

Результат: S – сумма – вещественный тип.

Вычисление суммы выполняем в функции *sum*. Рекуррентная формула, которая используется при вычислении суммы, следующая: $a_k = a_{k-1}(x^2)/$ $/((2k-1)(2k)$. Граничное условие для рекурсии: $a <$ eps.





Пример 3

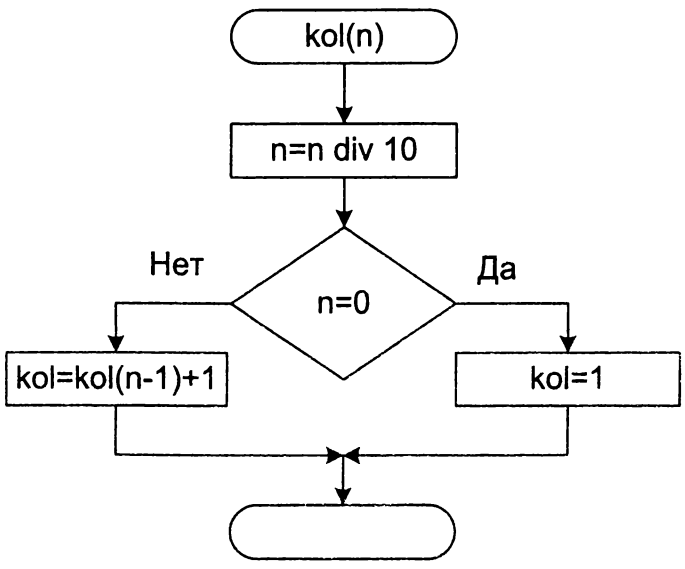
Подсчитать количество цифр в заданном натуральном числе.

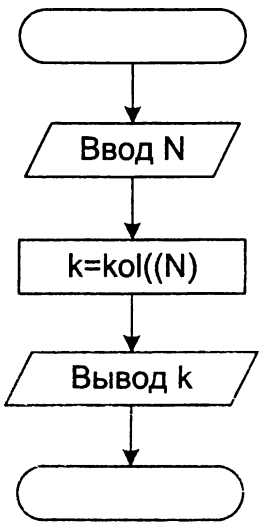
Исходные данные: N – заданное число – целый тип.

Результат: k – количество цифр – целый тип.

В рекурсивной функции для уменьшения количества цифр заданное число делится на 10. Деление прекращается, когда результатом деления будет ноль – это граничное условие для рекурсии.

Тестовый пример: при $N = 5674$ $k = 4$.





Приведем пример с рекурсивной процедурой.

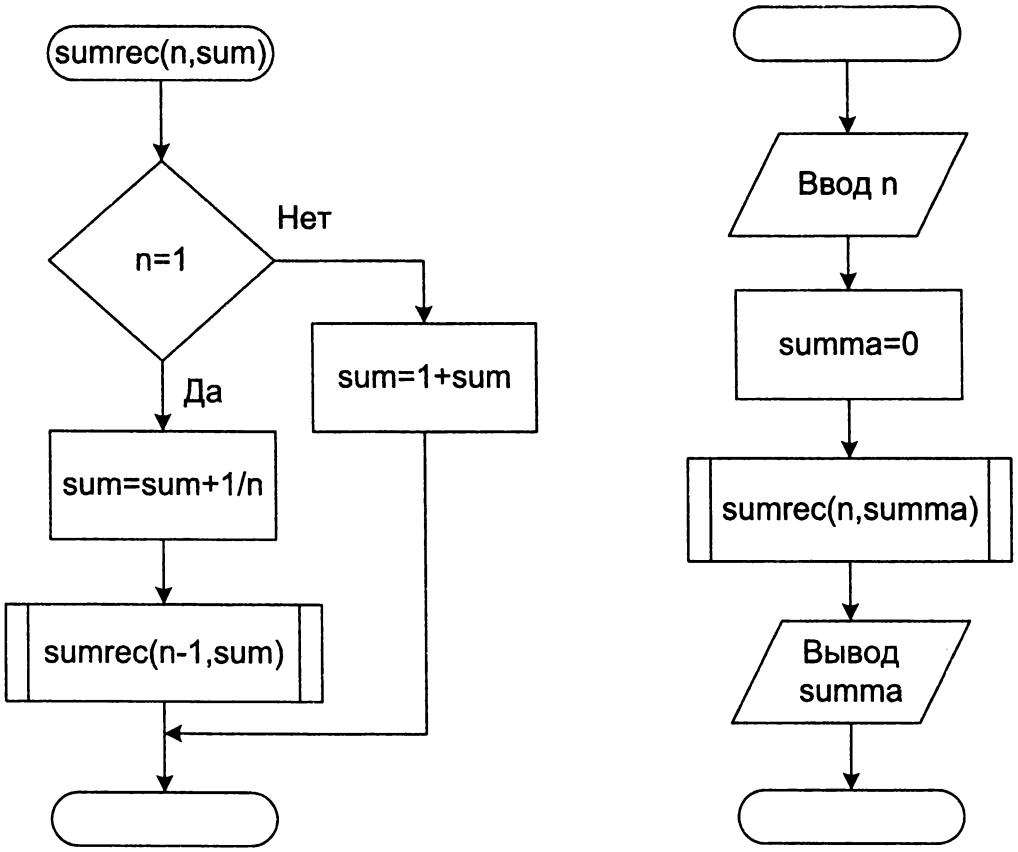
Пример 4

Вычислить $\sum_{k=1}^{n}\frac{1}{i}$ n членов ряда.

Исходные данные: n – количество слагаемых – целый тип.

Результат: summa – сумма ряда – целый тип.

Тестовый пример: при $n = 5$ summa = 2,2833.



Контрольные вопросы

1. Какая подпрограмма называется рекурсивной?
2. Почему в рекурсивной подпрограмме отсутствуют операторы цикла?
3. Может ли в рекурсивной подпрограмме отсутствовать развилка?
4. Почему в примере 2 отсутствует операция $k = k + 1$?
5. Как будет работать рекурсивная подпрограмма в примере 2 при eps = 0,1, $x = 1$?
6. Как будет работать рекурсивная подпрограмма в примере 3 при $k = 333$?

7. В примере 4 укажите параметры-значения и параметры-переменные в процедуре *sumrec*.

Задания для лабораторной работы

1. Найти сумму цифр заданного натурального числа.

2. Написать программу вычисления целой степени любого вещественного числа.

3. Составить программу для нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке.

4. Даны неотрицательные целые числа n, m. Вычислить A (n, m), где

$$A(n,m)=\begin{cases} m+1, \text{если } n=0, \\ A(n-1,1), \text{если } n\neq 0,\ m=0, \\ A(n-1,\ A(,m-1)), \text{ если } n>0,\ m>0. \end{cases}$$

5. Создать программу для вычисления $\sqrt[k]{a}$.

Для этого сначала нужно вычислить элементы числовой последовательности $x_0=a$: $x_i=\frac{k-1}{k}x_{i-1}+\frac{a}{kx_{i-1}^{k-1}}$, $i=1,2,\ldots$, и найти первое значение x_n, для которого $|x_n{}^k-a|<10^{-4}$.

6. Создать логическую функцию, которая возвращает *True*, если ее аргумент – простое число.

7. Даны действительные числа x и ε ($x\neq 0$, $\varepsilon>0$). Вычислить $\sum_{k=0}^{\infty}\frac{(-1)^k}{((k+1)!)^2}\left(\frac{x}{2}\right)^{2(k+1)}$ с точностью ε и указать количество учтенных слагаемых.

Вычисление выполнить через рекурсию.

8. Даны действительные числа x и ε ($x\neq 0$, $\varepsilon>0$). Вычислить $\sum_{k=0}^{\infty}\frac{(-1)^{k+1}x^{2k-1}}{(2k-1)!(2k+1)!}$ с точностью ε и указать количество учтенных слагаемых.

Вычисление выполнить через рекурсию.

9. Дано действительное число ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону:

$$a_n = \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \cdot \ldots \cdot \left(1 - \frac{1}{n+1}\right).$$

С помощью рекурсии вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

10. Дано действительное число ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону:

$$a_n = \left(1 - \frac{1}{2!}\right) \cdot \left(1 + \frac{1}{3!}\right) \cdot \ldots \cdot \left(1 - \frac{(-1)^n}{(n+1)!}\right).$$

С помощью рекурсии вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

11. Даны действительные числа x и ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону: $a_1 = x$; далее для n =2, 3, … выполнено

$$a_n = \frac{2}{a_{n-1}} + \frac{x}{4 + a_{n-1}^2}.$$

С помощью рекурсии вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

12. Даны действительные числа x и ε (ε > 0). Последовательность a_1, a_2, … образована по следующему закону: $a_1 = x$; далее для n = 2, 3, … выполнено

$$a_n = \frac{1}{\sqrt{\left|4a_{n-1}^2 - 2x + 1\right|}}.$$

С помощью рекурсии вычислить сумму первых членов a_n ($n \geqslant 2$), для которых выполнено условие $|a_n - a_{n-1}| < \varepsilon$.

Лабораторная работа 10. Стандартные процедуры и функции работы со строками

Теория

Строка – это последовательность символов. При использовании в выражениях строка заключается в апострофы. Количество символов в строке зависит от версии языка *Pascal*. В *Turbo Pascal* длина строки может динамически изменяться от 0 до 255. Для определения данных строкового типа используется идентификатор *string*, за которым следует заключенное в квадратные скобки значение максимально допустимой длины строки данного типа. Если это значение не указывается, то по умолчанию длина строки равна 255 байтам.

Для определения объема памяти, требуемой для размещения строки, в байтах к значению ее максимальной длины прибавляется 1. Дополнительный байт расположен в самом начале строки (имеет нулевой номер) и содержит значение текущей длины строки.

В *Object Pascal* для строки может быть выделено до 2 Гбайт. Конец строки определяется так называемым нулевым символом.

Выражения, в которых операндами служат строковые данные, называются *строковыми*. Они состоят из строковых констант, переменных, указателей функций и знаков операций. Над строковыми данными допустимы операция сцепления и операции отношения.

Операция сцепления (+) применяется для сцепления нескольких строк в одну результирующую строку. Например, результатом выражения 'А'+'Б'+'В'+'Г'+'Д' будет строка 'АБВГД'.

Операции отношения (=, <>, >, <, ⩾, ⩽) проводят сравнение двух строковых операндов и имеют более низкий приоритет, чем операции сцепления, т. е. вначале всегда выполняются все операции сцепления, если они присутствуют, и лишь потом реализуются операции отношения.

Сравнение строк производится слева направо до первого несовпадающего символа. Большей считается строка, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией. Результат выполнения операций отношения над строковыми операндами всегда имеет булевский тип и принимает значение *True*, если выражение истинно, и *False*, если выражение ложно. Если строки имеют

различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки считаются равными, если они полностью совпадают по длине и содержат одни и те же символы.

Например, 'akkord'>'AKKORD' равно True.

К отдельным символам строки можно обратиться по номеру (индексу) данного символа в строке. Индекс определяется выражением целочисленного типа, которое записывается в квадратных скобках сразу за идентификатором строковой переменной или константы.

Для обработки строковых данных используются приведенные в табл. 5 и табл. 6 стандартные функции и процедуры.

Таблица 5

Стандартные функции,
используемые для обработки строковых данных

Функция	Тип результата	Описание	Примеры
Length (*строка*)	integer	Определяет количество символов в строке	Length (‘12345’) Результат = 5
Copy (*строка, начальная позиция, количество символов*)	string	Копирует заданное количество символов с указанной позиции	Copy (‘ABCDE’, 2, 3) Результат =’BCD’
Concat (*строка 1, строка 2, ...строка n*)	string	Соединяет указанные строки	Concat (‘123’, ‘567’, ’1’) Результат = ‘1235671’
Pos (*подстрока, строка*)	integer	Определяет позицию, с которой подстрока входит в строку; если подстрока не содержится в строке, результат равен нулю	Pos ('de','abcdef’) Результат = 4
UpCase (*символ*)	char	Преобразует строчную букву в прописную	UpCase (‘n’) Результат ‘N’

Таблица 6

Стандартные процедуры,
используемые для обработки строковых данных

Процедура	Описание	Примеры
Delete (*строка, начальная позиция, количество символов*)	Удаляет из строки с указанной позиции заданное количество символов	Delete (St, 4, 2), где St='абвгде' Результат: St='абве'
Insert (*строка 1, строка 2, начальная позиция*)	Вставляет строку 1 в строку 2, начиная с заданной позиции	Insert ('123', St,4), где St='abcdef' Результат: St='abc123def'
Str (*число, строка*)	Преобразует число в строку. У числа может быть указан формат, аналогичный формату ввода	Str (1500:6, St), Результат: St=' 1500'
Val (*строка, число, код*)	Преобразует строку в число. Код равен нулю, если преобразование возможно, иначе код будет содержать номер позиции первого ошибочного символа, а значение числа не будет определено	Val ('145', N, Cod) Результат: N=145, Cod=0 Val ('34,5', N, Cod) Результат: N не определено, Cod=3

Примеры

Пример 1

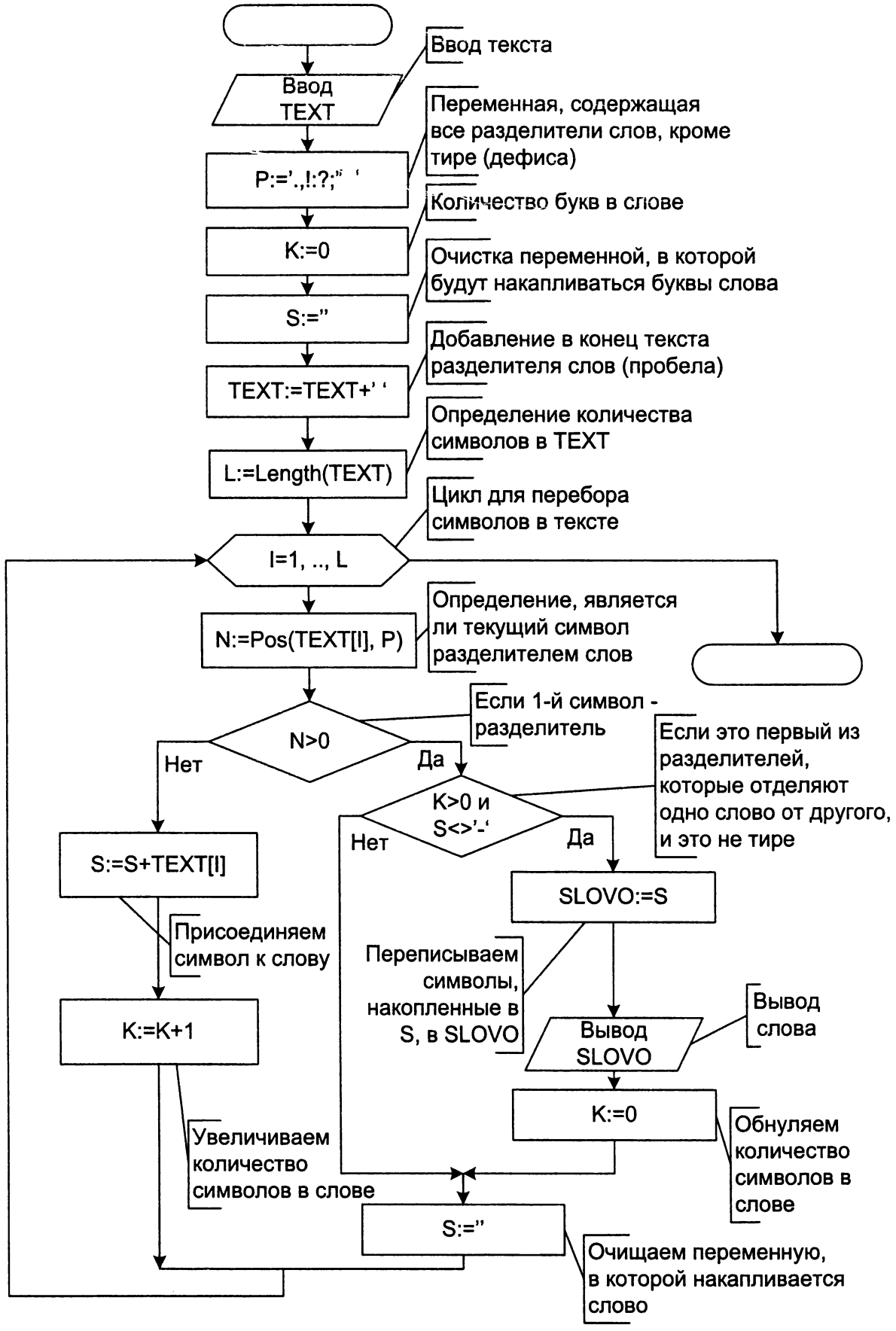
Дан текст. Напечатать все слова из 4 букв, содержащиеся в этом тексте.

Исходные данные: TEXT – строкового типа.

Результат: печатаются слова из 4 букв.

Просматриваются все символы текста. Если встречаются символы – разделители слов, считается, что достигнут конец слова. Определяется размер выделенного слова. Если длина слова равна 4, это слово печатается.

Тестовый пример: при TEXT='Сумма трех целых чисел' вывод на печать: «Сумма», «трех».


Ввод текста
Ввод TEXT
Переменная, содержащая все разделители слов, кроме тире (дефиса)
P:='.,!:?;" '
Количество букв в слове
K:=0
Очистка переменной, в которой будут накапливаться буквы слова
S:=''
Добавление в конец текста разделителя слов (пробела)
TEXT:=TEXT+' '
Определение количества символов в TEXT
L:=Length(TEXT)
Цикл для перебора символов в тексте
I=1, .., L
Определение, является ли текущий символ разделителем слов
N:=Pos(TEXT[I], P)
Если 1-й символ - разделитель
N>0
Нет
Да
Если это первый из разделителей, которые отделяют одно слово от другого, и это не тире
K>0 и S<>'-'
Нет
Да
S:=S+TEXT[I]
SLOVO:=S
Присоединяем символ к слову
Переписываем символы, накопленные в S, в SLOVO
Вывод слова
K:=K+1
Вывод SLOVO
K:=0
Обнуляем количество символов в слове
Увеличиваем количество символов в слове
S:=''
Очищаем переменную, в которой накапливается слово

Пример 2

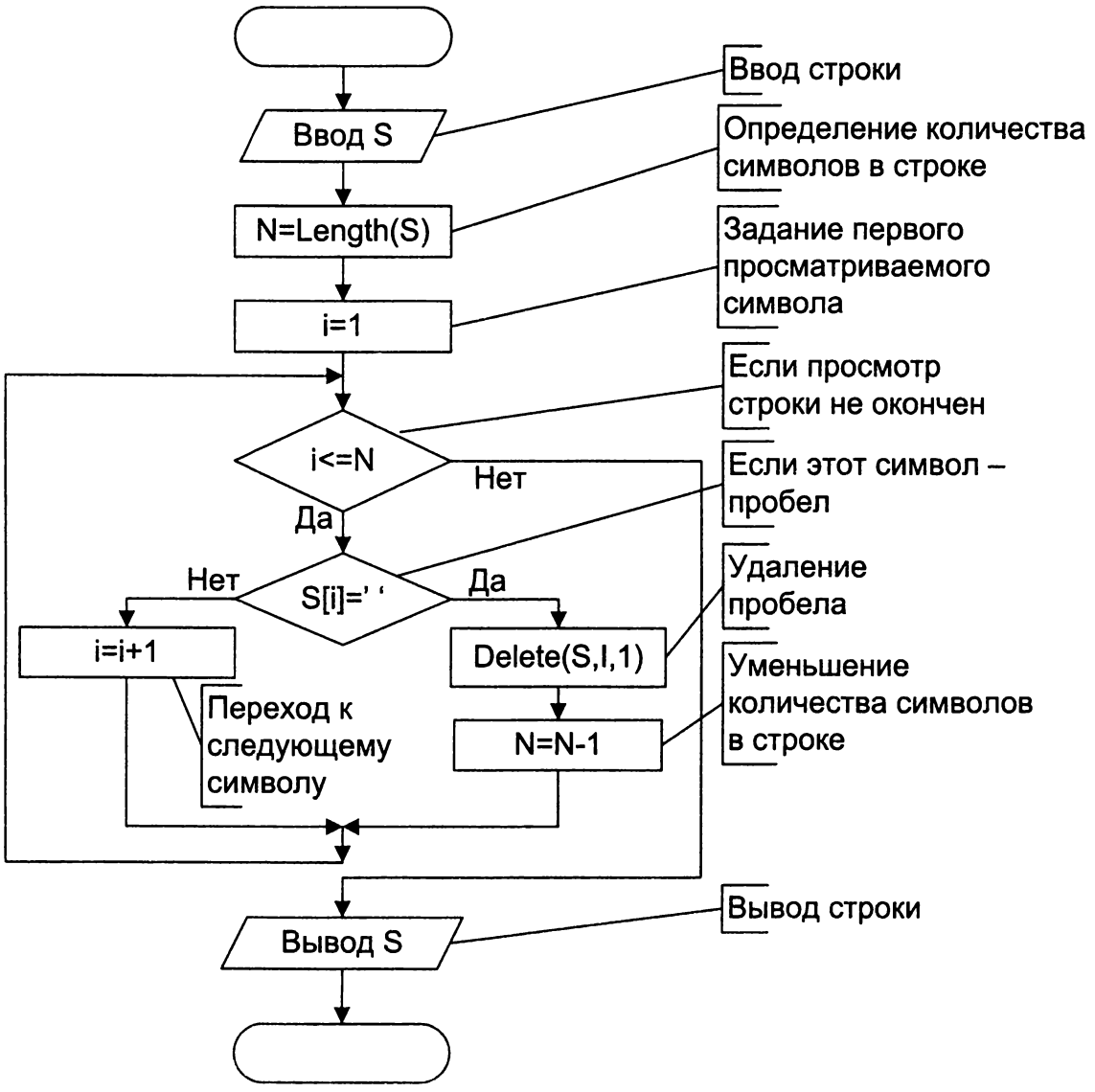
Дан текст. Удалить в этом тесте все пробелы.

Исходные данные: строка *S* – типа string.

Результат: строка *S* без пробелов.

Следует обратить внимание на то, что после удаления символа индекс символа *i* не изменяется из-за сдвига символов после удаления.

Тестовый пример: при *S* = 'Разделение строки на отдельные слова' конечный результат *S* = 'Разделениестрокинаотдельныеслова'.



Контрольные вопросы

1. Какой операцией можно заменить вычисление N:=Lengh (Text)?
2. Сколько байт будет выделено для строки, описанной как S: string [10]?

3. Сколько символов можно записать в строке, описанной как St: string?

4. Каков результат процедуры Insert (‘+‘, A,4), где A=’Пример’?

5. Чему равен результат выражения ‘23’+’77’?

Задания для лабораторной работы

1. Дан текст. Сколько слов в тексте начинаются и заканчиваются на одну и ту же букву?

2. Дана строка, содержащая фамилию, имя, отчество в любом регистре. Получить фамилию и инициалы. Первая буква фамилии заглавная, остальные строчные. Инициалы – заглавные буквы.

3. Проверить, одинаковое ли число открывающихся и закрывающихся скобок в данной строке. Причем каждая открывающаяся скобка должна предшествовать закрывающейся.

4. Дан текст, в котором после знаков препинания могут отсутствовать пробелы. Вставить пробелы после каждого знака препинания, а если знак – тире, то и перед ним. Каждое слово, стоящее после точки, написать с большой буквы.

5. Дан текст. Найти самое длинное слово в тексте.

6. Дан текст, содержащий полную информацию о фирме. Вывести расчетный счет фирмы. Начало счета может начинаться со слов «расчетный счет:», «р/с:» или «№ р/с:». Количество цифр в расчетном счете не известно.

7. Дан текст, содержащий полное название организации. Получить аббревиатуру этой организации. В названии могут встречаться слова, пишущиеся с маленькой буквы, но в аббревиатуре все буквы должны быть заглавными.

8. Дан текст, содержащий направление и цену туристической путевки, например, «Египет – 45000, Сочи – 30000». По введенному направлению вывести цену путевки. Путевки с одним направлением могут встречаться несколько раз.

9. В записке каждое слово зашифровано – записано наоборот. Расшифровать сообщение.

10. Дан текст, содержащий фамилию и дату рождения человека в виде «Иванов А. М. 24.5.1970». Преобразовать этот текст к виду «Иванов А. М. 24 года».

11. Дан текст. Удалить лишние пробелы в тексте. Пробел считается лишним, если он стоит в начале строки, в конце строки и если он следует за пробелом.

12. Дана строка, содержащая название товара и его цену. Например: «Конфеты шоколадные “Перезвоны” 70 р.», «Хлеб Чусовской 7 р. 30 к.» или «Колбаса 103,40». Разнести в разные строки название товара и его цену. Цену преобразовать в число.

Лабораторная работа 11. Модули

Теория

Модуль – это набор ресурсов (функций, процедур, констант, переменных, типов и т. д.), разрабатываемых и хранимых отдельно от использующих их программ.

Модуль состоит из следующих частей:

1. Заголовок.
2. Раздел интерфейса.
3. Раздел реализации.
4. Конец модуля.

В заголовке определяется имя модуля. Это имя в дальнейшем используется при ссылке на модуль в предложении *uses*. Перед именем модуля стоит служебное слово «Unit»:

Unit (имя модуля);

Интерфейсная часть начинается со слова «interface». В этой части объявляются только те константы, типы, переменные, процедуры и функции, которые являются глобальными, т. е. доступны основной программе. В интерфейсной части заголовки процедур и функций лишь перечисляются.

Раздел реализации начинается со слова «implementation». В этом разделе приведены описания всех глобальных процедур и функций. В нем же описываются все константы, типы переменных, процедуры и функции, которые являются глобальными для модуля, но локальными по отношению к основной программе (т. е. основная программа их не видит).

Модуль заканчивается словом «end» с точкой: *end.*

Примеры

Пример 1

Создать модуль, содержащий функции возведения вещественного числа в целую степень, извлечения корня целой степени и возведение числа в произвольную степень.

Использовать этот модуль для вычисления z. $z = |x|^a - y^3$, где $x = \sqrt[4]{a^2 + b^4}$, $y = 2a - \sqrt{b}$ для произвольных a и b.

Модуль:

```
Unit Power;
Interface
{Функция возведения в целую степень n вещественного числа x}
Function IntPower (x: real; n: integer): real;
{Функция извлечения целого корня n вещественного числа x}
Function Root (x: real: n: integer): real;
{Функция возведения в произвольную степень y вещественного положительного числа x}
Function RealPower (x, y: real): real;
Implementation
Function IntPower (x: real; n: integer): real;
var P: real; i: integer;
begin
P:=1;
For i:=1 to n do
P:=P*x;
IntPower:=P;
end;
Function Root (x: real; n: integer): real;
y: real;
begin
y:=x;
repeat
y:=(n-1)/n*y+x/(n*IntPower(y, n-1));
until abs (IntPower (y, n)–x)<1.0E-4;
Root:=y;
end;
Function RealPower (x, y: real): real;
begin
```

```
RealPower:=exp (y*ln (x));
end;
end.
```

Основная программа:

```
program Powerz;
Uses Power;
var a, b, x, y, z: real;
begin
write ('a, b=');
readln (a, b);
y:=2*a-sqrt (b);
x:=Root (sqr (a)+IntRower (b,4)), 4);
z:=RealPower (abs (x), a) – IntRower (y, 3);
writeln ('z=', z:10:4);
readln;
end.
```

Контрольные вопросы

1. Как подключить к программе модуль?
2. С какого слова начинается текст модуля?
3. Для чего предназначен раздел *implementation*?
4. Какие подпрограммы и данные можно не описывать в разделе *interface*?
5. Сколько модулей можно подключать к программе?

Задания для лабораторной работы

1. Создать модуль и разработать перечисленные ниже функции, затем выполнить задание с текстом. Стандартные функции не использовать.

Функции: MaxText – поиск самого длинного слова в тексте; NewPos – определение вхождения подстроки с заданной позиции; Convers – переписывание заданного текста наоборот; MyCopy – выделение с заданной позиции заданного количества символов; MyLen – определение количества символов в строке.

Задание. Дан текст. Найти самое длинное слово в тексте и все слова, равные по длине этому слову, переписать наоборот.

2. Создать модуль и разработать перечисленные ниже функции, затем выполнить задание с текстом.

Функции: WordN – выделение N-го слова; MyMove – перенос заданной подстроки в заданную позицию; CountWord – определение количества слов в тексте; MyInsert – вставка подстроки с заданной позиции.

Задание. Дан текст. Попарно переставить слова в этом тексте.

3. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: CountWord – определение количества слов в тексте; WordN – выделение N-го слова; DeleteAll – удаление всех вхождений заданной подстроки; DivEnd – разделение строки на две подстроки по последнему вхождению сочетания символов.

Задания: а) дан текст. Удалить в этом тексте все слова, равные 4 символам; б) текст дан как описание товара и его цена. Цена отделяется от описания знаком «–». Разнести по подстрокам описание товара от его цену.

4. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: MoveEnd – перенос заданной подстроки в конец; MyPos – определение первого вхождения подстроки; WordN – выделение N-го слова; CountWord – определение количества слов в тексте; Sovpad – проверка совпадения двух строк без учета регистров, пробелов и знаков препинания.

Задания: а) даны две строки, содержащие слова, написанные в разных регистрах. Найти в первой строке слова, совпадающие со словами из второй строки; б) поменять местами в тексте первое и последнее слово.

5. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: NewCopy – выделение строки с заданного начального символа до заданного конечного символа; NewDelete – удаление подстроки с начальной позиции до конечной; MyPos – определение первого вхождения подстроки; MyDown – написание всех букв в строке строчными.

Задания: а) дан текст, содержащий фрагмент в круглых скобках. Переписать этот фрагмент в отдельную строку и сделать все буквы в этой строке строчными; б) из первоначального текста этот фрагмент удалить.

6. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: Inicial – преобразование текста с фамилией, именем, отчеством или фамилией и именем в текст с фамилией и инициалами, преобразование регистра; MyReplace – замена в строке одной подстроки на другую;

NewPos – определение вхождения подстроки с заданной позиции; CountStr – вычисление вхождений одной подстроки в другую; DeleteStr – удаление k-го вхождения заданной подстроки.

Задания: а) дан текст, содержащий в середине фамилию, имя, отчество, которым предшествует слово «уважаемый». Заменить имя и отчество на инициалы и записать их с большой буквы; б) в тексте несколько раз встречается знак «!». Удалить последний «!».

7. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: Abrev – создание аббревиатуры заданного текста; NewPos – определение вхождения подстроки с заданной позиции; MyReplace – замена в строке одной подстроки на другую; NewRight – выделение с конца строки, начиная с заданного символа.

Задания: а) дан адрес организации. Название организации заключено в кавычки. Заменить название организации на аббревиатуру; б) в конце текста указан город, чему предшествует обозначение «г.». Выделить название города в отдельную переменную.

8. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: MyNumber – запись целого числа (до миллиона) числительными; MyLeft – выделение сначала заданного количества символов; MyDelete – удаление подстроки (заданного количества символов) с заданной позиции.

Задания: а) дан текст, который содержит число (не более 6 цифр). Записать это число прописью; б) дан адрес. В начале адреса указан почтовый индекс. Выделить его в отдельную строку, а в адресе индекс удалить.

9. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: MyVal – преобразование строки в целое число; NewCopy – выделение строки с заданного начального символа до заданного конечного символа; MyPos – определение первого вхождения подстроки; MyLen – определение количества символов в строке.

Задания: а) дана строка, содержащая название товара и его цену. Например: «Конфеты шоколадные “Перезвоны” 70 р.», «Хлеб Чусовской 7 р. 30 к.» или «Колбаса 103,40». Разделить название товара и его цену, записав последнее в переменную вещественного типа; б) в названии товара все буквы сделать заглавными.

10. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: MyPos – определение первого вхождения подстроки; ReplaceAll – замена в строке всех вхождений одной подстроки на другую; Rubl – преобразование числа в строку денежного формата; MyPropose – запись всех слов в строке с большой буквы, остальные буквы – строчные.

Задания: а) дан юридический адрес предприятия, указан его расчетный счет, перед которым стоит пометка «р/с» или «расчетный счет». Заменить один расчетный счет на другой; б) дана строка с фамилией сотрудника. В виде числа указан его оклад. Объединить эти данные в одну строку. Фамилию, имя, отчество начинать с заглавных букв.

11. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: ReplaceAll – замена в строке всех вхождений одной подстроки на другую; NewPropose (строка) – запись первого слова с большой буквы, остальных слов – со строчной буквы; TrimAll – удаление всех лишних пробелов; MyStr – преобразование целого числа в строку.

Задания: а) дан текст. Заменить все слова «тебя» в этом тексте на «Вас»; б) дана строка, содержащая фамилию ученика и класс, а также целое число – отметку. Объединить эти данные в строку. Фамилию ученика написать с заглавной буквы, удалив лишние пробелы.

12. Создать модуль и разработать перечисленные ниже функции, затем выполнить задания с текстом.

Функции: MoveBeg – перенос заданной подстроки в начало; CountWord – определение количества слов в тексте; WordN – выделение N-го слова; DivBegin – разделение строки на две подстроки по первому вхождению сочетания символов.

Задания: а) дан текст. Переписать слова в этом тексте в обратном порядке; б) дан текст: фамилия ученика, предмет и отметка. Отметка отделена словом «оценка». Получить две строки: одна – фамилия и предмет, другая – отметка.

Тема 5. АЛГОРИТМЫ РАБОТЫ СО СТРУКТУРИРОВАННЫМИ ТИПАМИ ДАННЫХ

Наряду с простыми типами данных, рассмотренными в предыдущих темах, существуют ***структурированные***. Переменные этих типов представляют собой совокупность связанных данных и правил, определяющих их организацию и способ доступа к элементам данных. Выбором той или иной структуры данных определяется и алгоритм их обработки, а значит, и то, насколько эта обработка будет эффективна.

В языке *Pascal* определены следующие стандартные структурированные типы данных:

- массив;
- запись;
- строка;
- множество.

Об алгоритмах работы со строками мы уже говорили. В данной теме будут рассмотрены алгоритмы работы с массивами и записями.

Лабораторная работа 12. Стандартные алгоритмы работы с одномерными массивами

Теория

Массив представляет собой совокупность пронумерованных однотипных значений, имеющих общее имя. Элементы массива обозначаются переменными с индексами. Индексы записывают в квадратных скобках после имени массива. Например, T [*l*], T [5], T [*i*], H [1981,9], H [*i*, *j*] и т. п.

Массив, хранящий линейную таблицу, называется *одномерным*. Тип элементов массива называется его *базовым* типом.

Массив описывается в разделе описания переменных в следующей форме:

Var ИмяМассива: Array [*ТипИндекса*] Of *ТипЭлемента*

Чаще всего в качестве типа индекса употребляется интервальный тип. Например, Var T: Array [1..12] Of Real;

Описание массива определяет, во-первых, размещение массива в памяти, во-вторых, правила его дальнейшего употребления в программе. Последовательные элементы массива располагаются в последовательных ячейках памяти (T [1], T [2] и т. д.), причем значения индекса не должны выхо-

дить из диапазона от 1 до 12. В качестве индекса может употребляться любое выражение соответствующего типа. Например, T [*i* + *j*], T [*m div* 2].

Тип индекса может быть любым скалярным порядковым типом, кроме *integer*.

Часто структурированному типу присваивается имя в разделе типов, которое затем используется в разделе описания переменных:

```
Type
Masl=Array [1..100] Of Integer;
Mas2=Array [–10..10] Of Char;
Var Num: Masl; Sim: Mas2;
```

В *Turbo Pascal* существуют только статические массивы, а это означает, что при описании массива нужно точно указать границы его индексов. Изменение размеров массива происходит через изменение в тексте программы и повторную компиляцию. Для упрощения таких изменений удобно определять индексные параметры в разделе констант:

```
Const N=10;
Var Mas: Array [1..N] Of Integer;
```

Теперь для изменения размеров массива *Mas* и всех операторов программы, связанных с этими размерами, достаточно отредактировать только одну строку в программе – раздел констант.

Другой вариант использования в программе массива с изменяющимся количеством элементов – зарезервировать место для элементов массива «с запасом», а во время работы программы вводить количество используемых элементов.

Действия над массивом как единым целым допустимы лишь в двух случаях:

- при присваивании значений одного массива другому;
- при выполнении операции отношения «равно – не равно».

В обоих случаях массивы должны иметь один тип (тип индексов и тип элементов). Например, Var A, B: Array [1..10] Of Real;

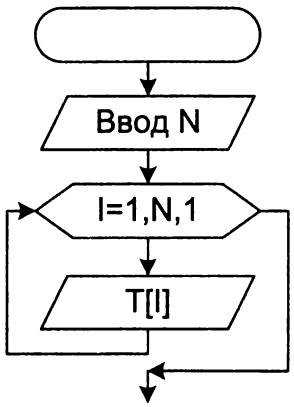
При выполнении операции присваивания A: =B все элементы массива A станут равны соответствующим элементам массива B.

Массив можно описать как типизированную константу. В этом случае начальные значения элементов массива задаются непосредственно в программе:

```
Type mas=array [1..7] of integer;
Const A: mas = (3, 6, -2, 4, 0, 5, -1);
```

В отличие от обычных констант значение типизированных может изменяться во время работы программы. Таким образом, описание массива через типизированные константы – его своеобразная инициализация.

Обработка массивов в программах производится покомпонентно, т. е. в цикле. Приведем блок-схему и программу ввода значений в массивы:




```
program Vvod;
var N, I: integer;
T: array [1..100] of real;
begin
WriteLn (‘Укажите кол-во элементов массива’);
ReadLn (N);
For I:=I To N Do
begin
WriteLn ('Введите T [‘, I,’]’);
ReadLn (T [I]);
end;
```

Здесь каждое следующее значение будет вводиться с новой строки с соответствующими комментариями. Для построчного ввода используется оператор *Read.*

Для ввода элементов массива можно создать процедуру. Чтобы использовать массив как параметр, его нужно описать как новый тип. Например, следующим образом:

```
Type
Msa=Array [1..20] Of Integer;
Var T: Mas,
N: Integer;
Procedure Vvod (Nmas: Integer; Var A: Mas);
Var I: Integer;
begin
```

```
For I:=1 To Nmas Do
Read (A [I]);
end;
```

Стоит обратить внимание на то, что массив определяется как параметр-переменная, так как в процедуре значение элементов массива изменяется.

В основной программе ввод будет иметь следующий вид:

```
begin
WriteLn ('Укажите кол-во элементов массива');
ReadLn (N);
WriteLn ('Введите элементы массива в строчку');
Vvod (N, T);
```

Аналогично в цикле по индексной переменной организуется вывод значений массива. Блок-схема этого алгоритма ничем не отличается от блок-схемы ввода, приведенной выше.

Пример программы вывода элементов массива:

```
For I:=I To N Do
WriteLn ('T [', I,']= ', T [I]);
```

Эта программа выводит каждый элемент в отдельной строке.

Процедура вывода элементов массива в строчку будет иметь вид:

```
Procedure Vivod (Nmas: Integer, A: Mas);
Var I: Integer;
begin
For I:=1 To Nmas Do
Write (A [I]:5);
WriteLn;
end;
```

Обратите внимание, что при выводе массив описывается как параметр-значение, а не как параметр-переменная.

В дальнейшем ввод и вывод элементов массива будут выполняться в процедурах.

Примеры

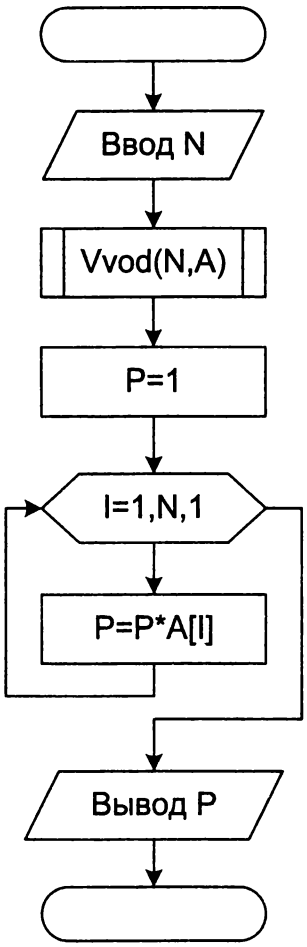
Пример 1

Вычислить произведение элементов массива.

Исходные данные: N элементов вещественного массива A, N – целый тип.

Результат: P – произведение элементов массива – целый тип.

Тестовый пример: при $N = 5$ элементы массива $A = 1, 3, 5, 3, 4$, $P = 180$.



Пример 2

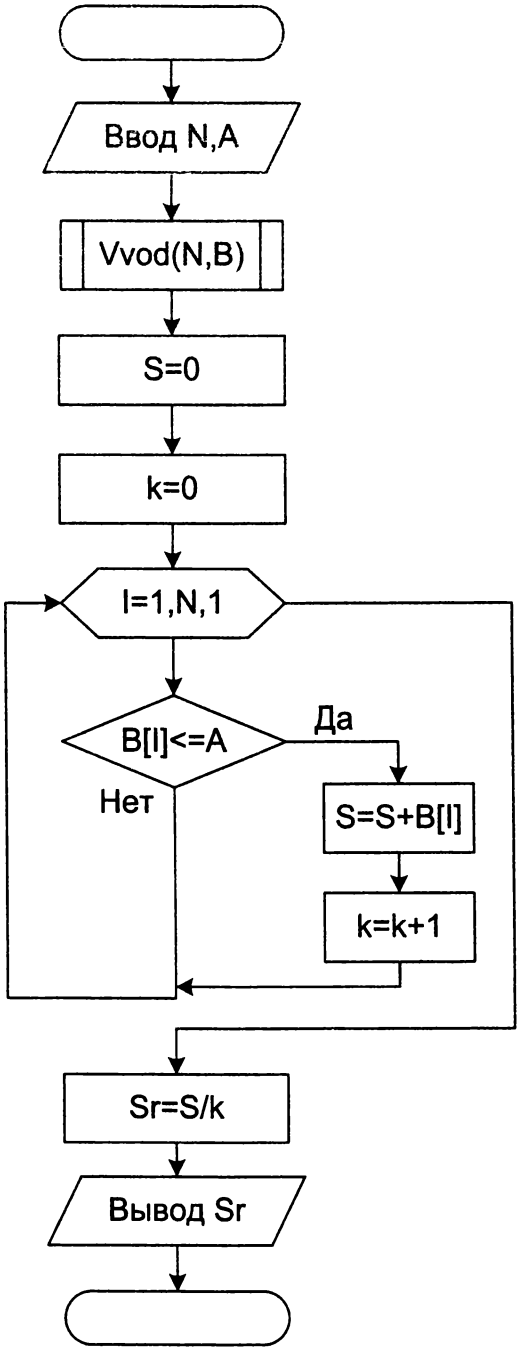
Найти среднее арифметическое значение элементов массива, не превышающих числа A.

Исходные данные: N элементов вещественного массива B, A – вещественное число.

Результат: Sr – среднее арифметическое значение элементов массива.

Промежуточные значения: I – индекс элементов массива; S – сумма элементов массива, не превышающих число A; k – количество элементов массива, не превышающих число A.

Тестовый пример: при $N = 7$, $A = 2$ элементы массива $B = 1, -3, 5, 4, -4, 6, 2$, Sr = –1.



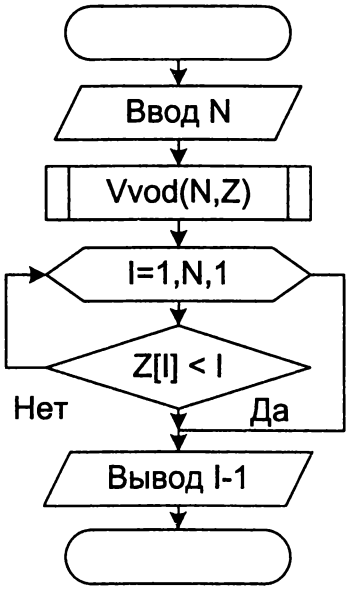
Пример 3

Определить, сколько элементов целочисленного массива стоит до первого элемента, значение которого меньше своего индекса.

Исходные данные: целочисленный массив Z из N элементов.

Результат: $I-1$ – количество элементов, стоящих до первого элемента, меньшего своего индекса.

Тестовый пример: при $N=7$ элементы массива $Z=1, 6, 15, 4, 0, 6, -1$, $I-1=3$.



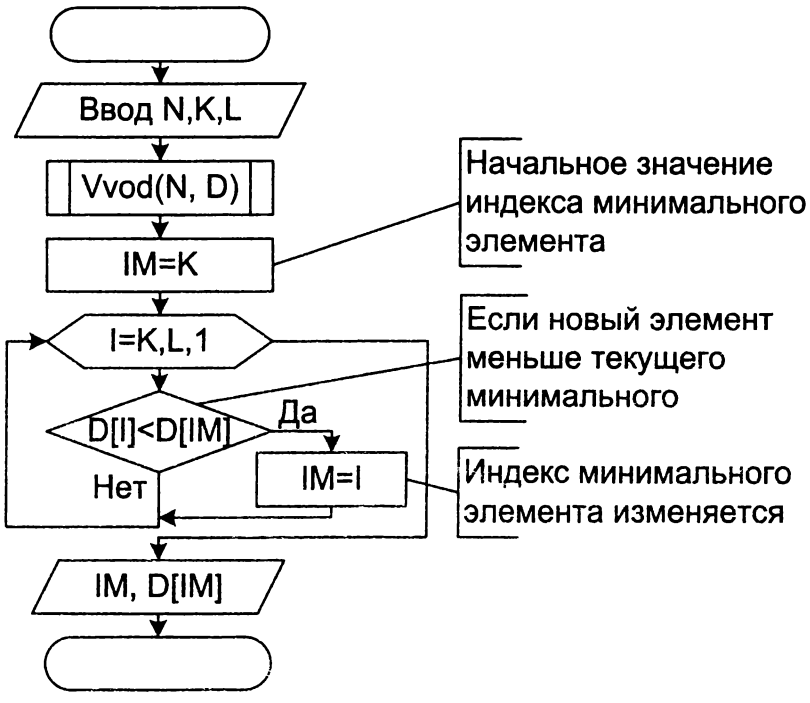
Пример 4

Найти минимальный элемент и его индекс среди элементов массива с номерами от *K* до *L*.

Исходные данные: целочисленный массив *D* из *N* элементов, *K* – начало поиска, *L* – конец поиска.

Результаты: IM – номер минимального элемента в заданном наборе элементов. Для минимального элемента дополнительная переменная не нужна, потому что его значение – это D [IM].

Тестовый пример: при *N* = 7 элементы массива D = 1, –3, 5, 4, –4, 6, 2 IM = 5, D [IM] = –4.



Пример 5

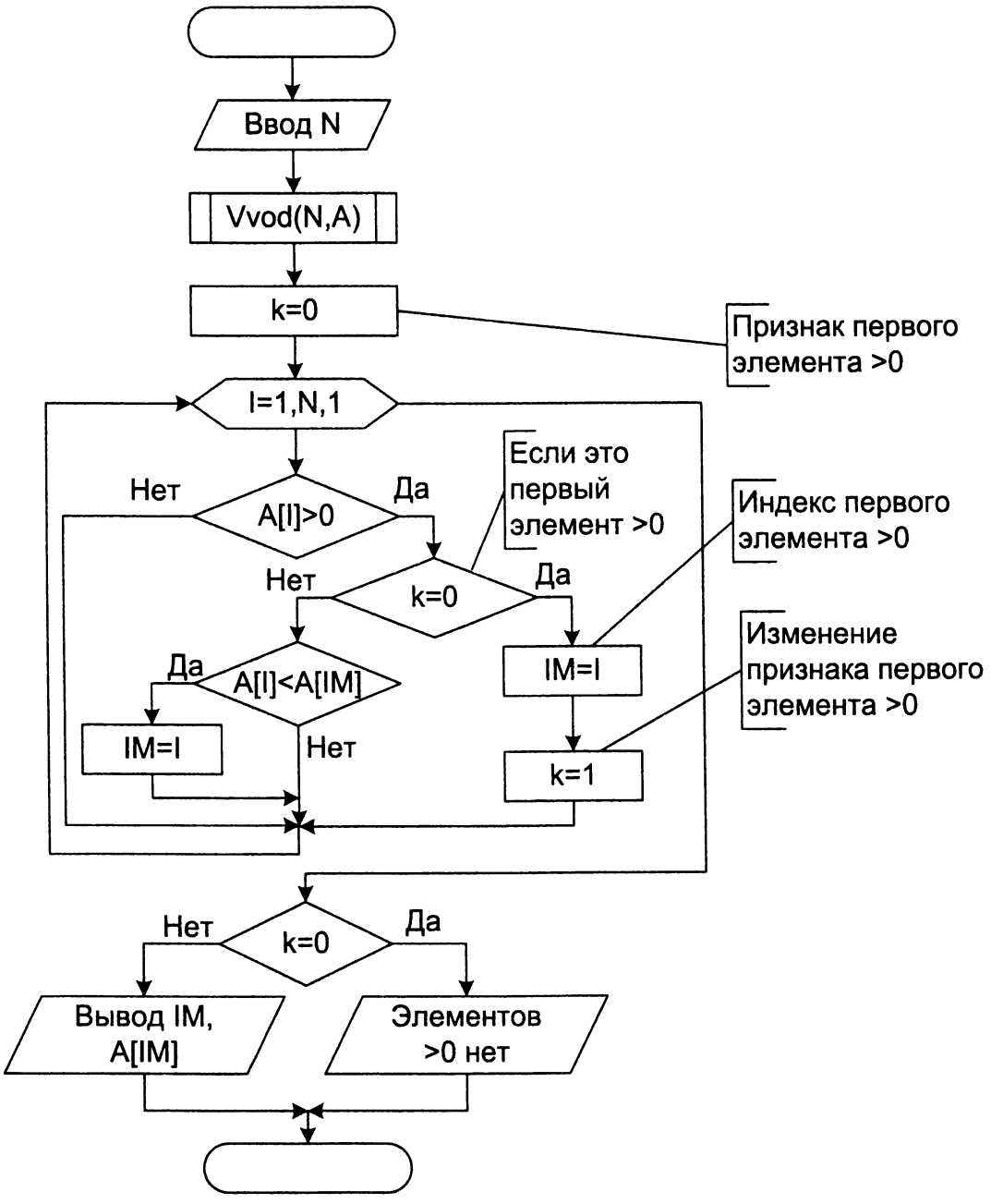
Определить номер и значение минимального элемента среди положительных элементов массива вещественных чисел.

Исходные данные: вещественный массив A из N элементов.

Результат: IM – номер минимального элемента.

Промежуточные значения: k определяет наличие положительного элемента в массиве. k равно нулю, пока не встретится положительный элемент.

Тестовый пример: при $N = 10$ и элементах –3, –5, 7, –2, 5, 2, –7, 5, 3, 6 IM = 6, $A[6] = 2$.



Пример 6

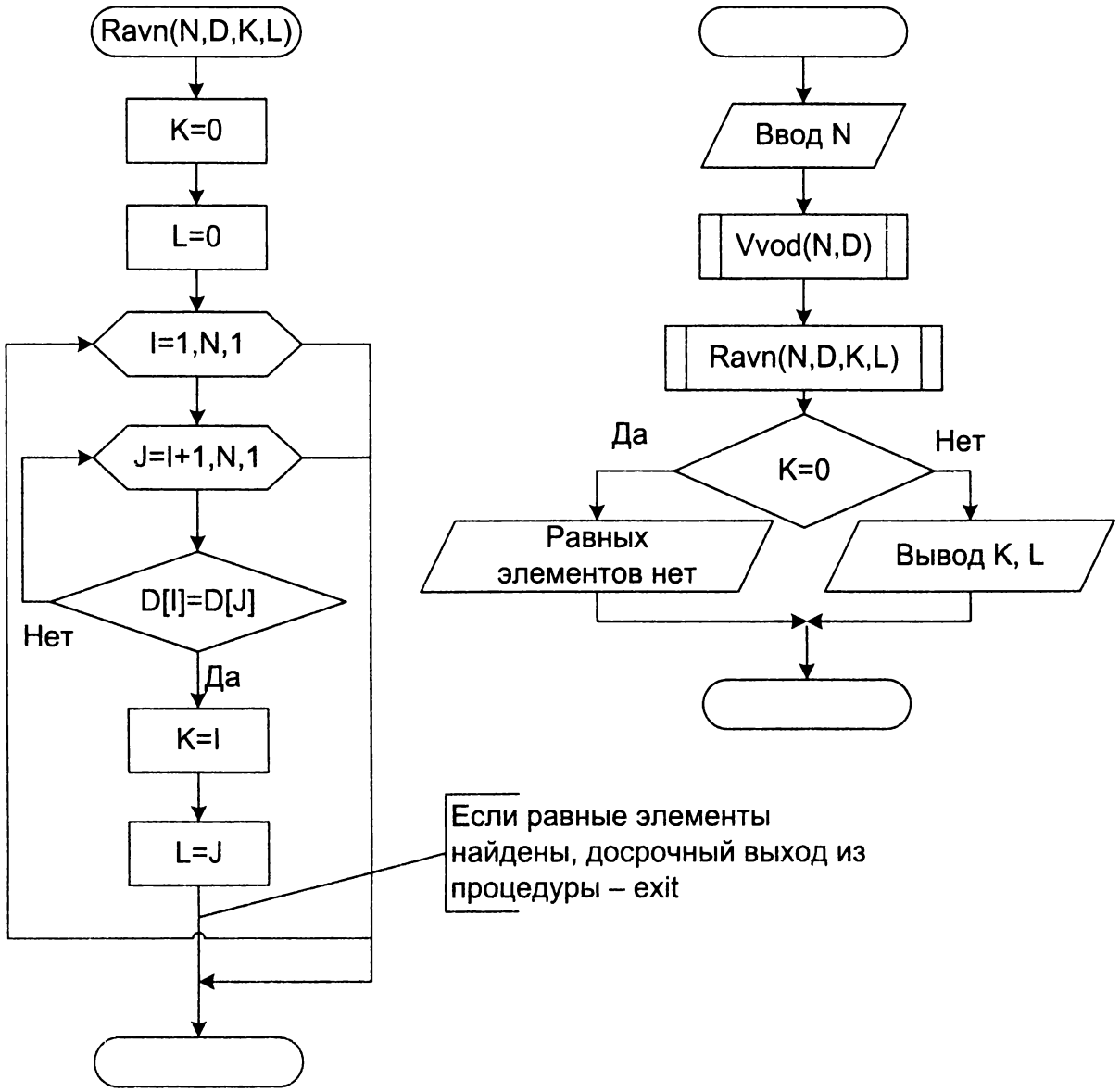
В целочисленном массиве найти номера двух первых равных элементов.

Исходные данные: целочисленный массив D из N элементов.

Результат: K, L – номера первых равных элементов.

В данном примере поиск равных элементов лучше вести в процедуре – в этом случае при нахождении равных элементов можно досрочно выйти из процедуры, а значит, сразу из двух циклов.

Тестовый пример: при $N = 7$ и элементах 4, 6, 3, 6, 3, 7, 1 $K = 2$, $L = 4$.



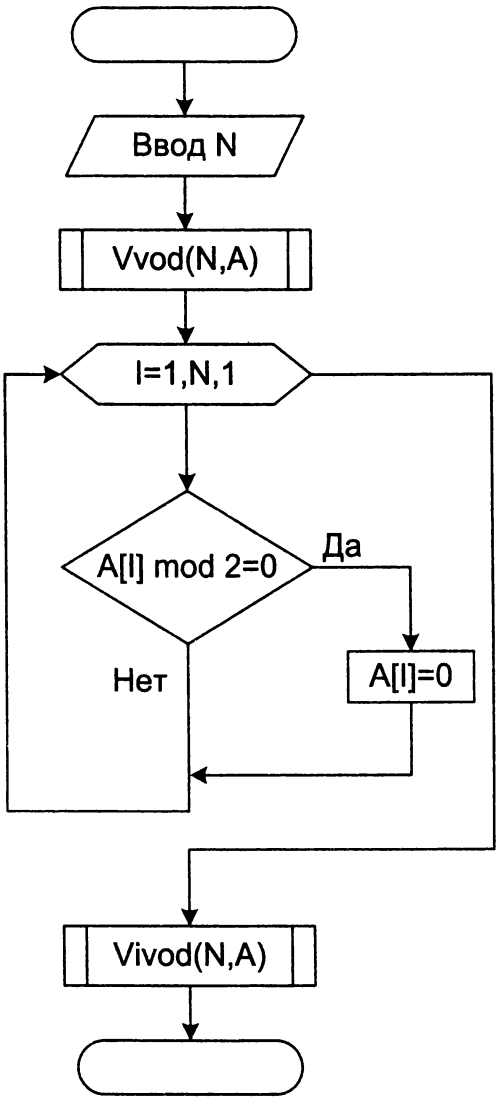
Пример 7

Заменить элементы, имеющие четное значение, на нуль.

Исходные данные: целочисленный массив A из N элементов.

Результат: преобразованный целочисленный массив A из N элементов.

Тестовый пример: при $N = 7$ и введенном массиве 4, 6, 7, 9, 3, 2, 1 преобразованный массив 0, 0, 7, 9, 3, 0, 1.



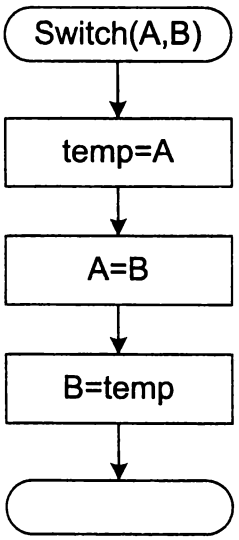
Пример 8

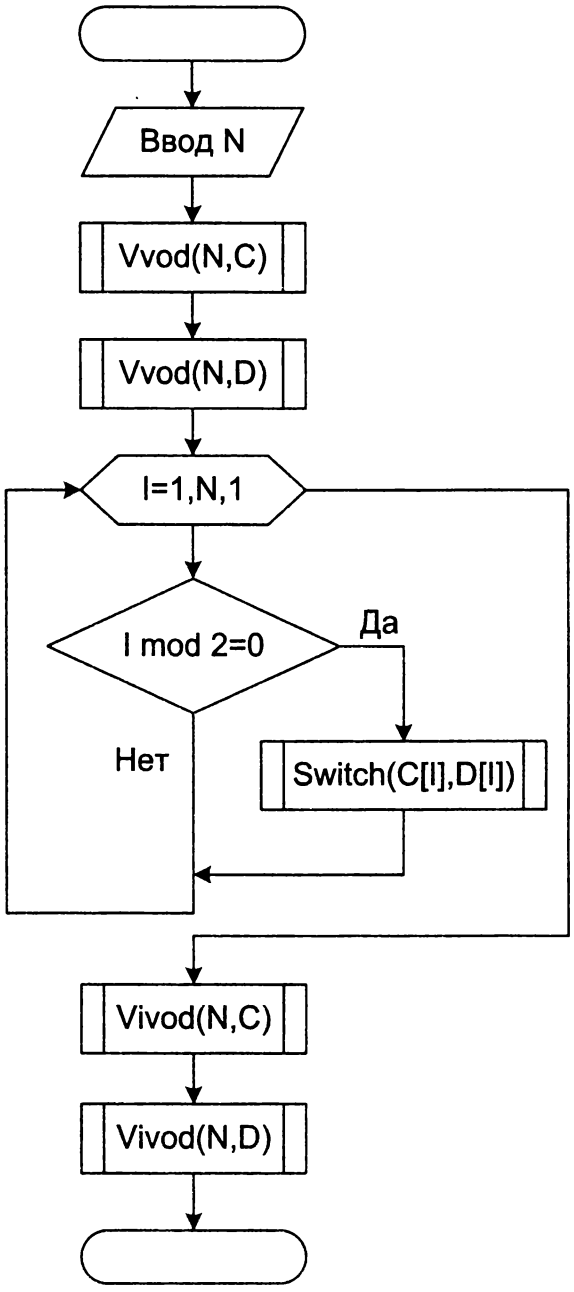
Переставить местами значения элементов двух массивов, имеющих четные индексы.

Исходные данные: вещественные массивы C и D из N элементов.

Результат: преобразованные вещественные массивы C и D из N элементов.

Тестовый пример: при $N = 5$ и введенных массивах C: 1, 2, 3, 4, 5 и D: –1, –2, –3, –4, –5 преобразованные массивы C: 1, –2, 3, –4, 5 и D: –1, 2, –3, 4, –5.





Пример 9

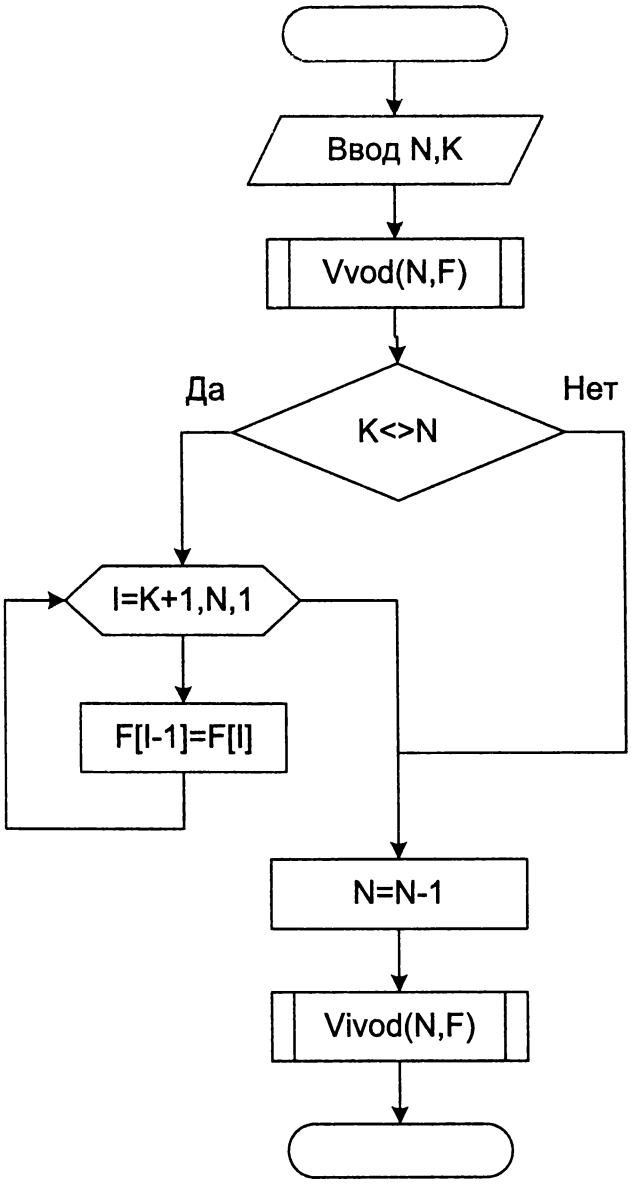
Удалить элемент с номером K.

Дано: вещественный массив F из N элементов, K – номер удаляемого элемента.

Результат: вещественный массив F из $N - 1$ элементов.

При удалении элементы массива сдвигаются влево.

Тестовый пример: при $N = 7$ и $K = 4$ ввод «5, 6, 4, 8, 1, 9, 2», вывод «5, 6, 4, 1, 9, 2», $N = 6$.



Пример 10

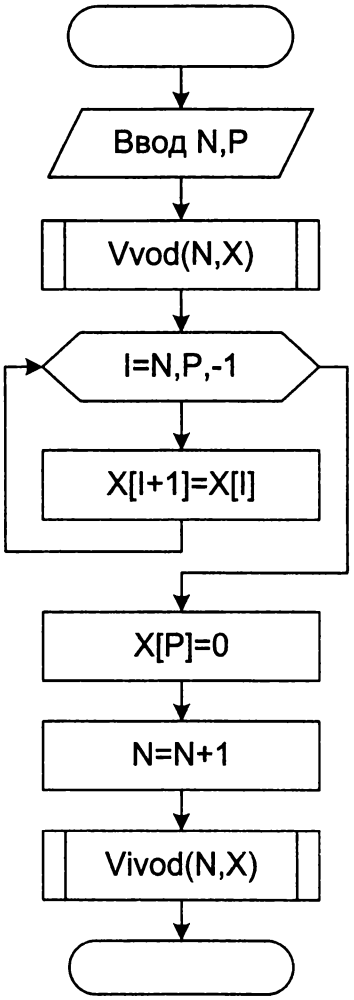
Вставить в массив после заданного элемента нуль.

Дано: целочисленный массив X из N элементов, P – номер элемента.

Результат: целочисленный массив X из $N + 1$ элементов.

При вставке элементы массива сдвигаются вправо. Чтобы не потерять элементы, сдвиг выполняется с конца.

Тестовый пример: при $N = 5$, $P = 3$ ввод «7, 3, 4, 8, 1», вывод «7, 3, 4, 0, 8, 1».



Пример 11

Создать программу, обеспечивающую работу следующих пунктов меню:

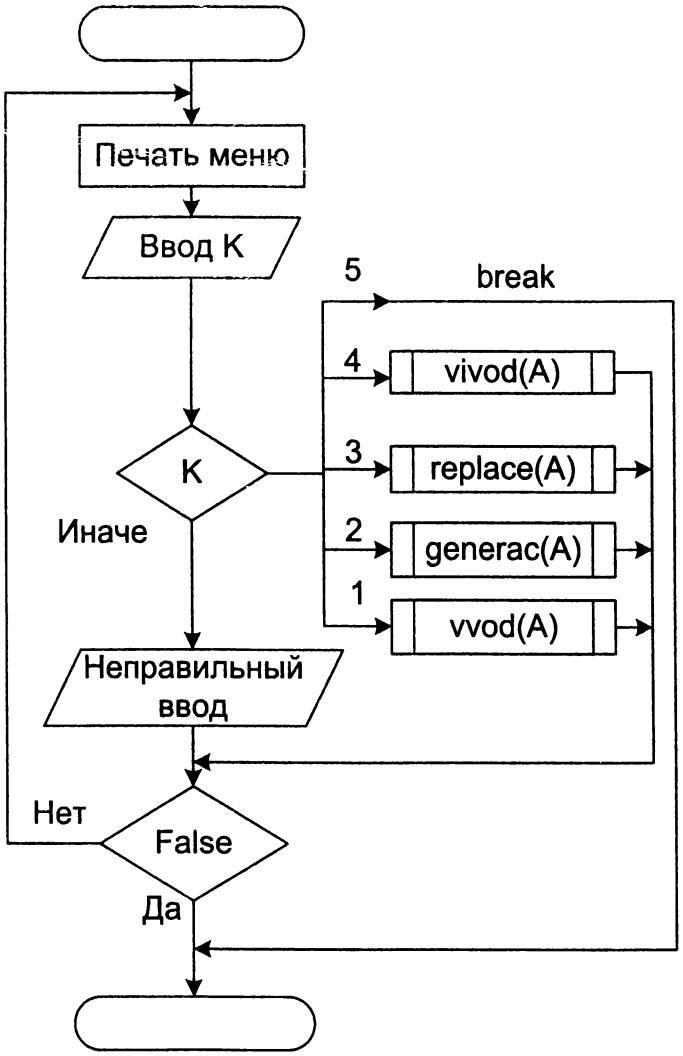
1. Ввод массива целых чисел из 20 элементов.
2. Генерация 20 элементов массива от –100 до 100.
3. Замена в массиве отрицательных элементов нулем.
4. Вывод элементов массива.
5. Конец работы.

Исходные данные: массив *A* из 20 целых чисел.

Результат: Полученный или преобразованный массив *A*.

В данном примере рассмотрена только основная программа выбора из меню. Блок-схема подпрограмм не приводится. Алгоритм содержит бесконечный цикл, чтобы после выполнения очередного пункта меню снова происходил возврат в меню. Но в этом случае потребовалось вводить

дополнительный пункт меню: *Выход из меню*, в данном случае – из цикла. Бесконечный цикл обеспечивается записью в условии *False*.




```
program menu;
Var K;
. . .

begin
 repeat
    writeln('1. Ввод массива');
    writeln('2. Генерация массива от
-100 до 100');
    writeln('3. Замена элементов
массива');
    writeln('4. Вывод массива');
    writeln('Выход из программы');
    writeln('Выберите пункт меню');
    writeln('Выберите пункт меню');
    readln(K);
    case K of
         1: vvod(A);
         2: generac(A);
         3: replace(A);
         4: vivid(A);
         5: break;
          else writeln('Неправильный
ввод');
     end;


end.
```

Контрольные вопросы

1. Какой тип данных не допускается для индекса?
2. Могут ли в описании массива в индексах содержаться переменные?
3. Что нужно сделать, если заранее количество элементов не известно?
4. Почему в примере 6 сравнивать элементы лучше в процедуре?
5. Почему в примере 10 цикл выполняется от N до 1?
6. Может ли массив являться параметром цикла? Если да, то что для этого нужно сделать?
7. Почему при вводе массива массив описывается как параметр-переменная, а при выводе – как параметр-значение?

Задания для лабораторной работы

I. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Вычисление среднего арифметического значения элементов массива и замена положительных элементов массива целой частью от среднего арифметического значения.

4. Конец работы.

II. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Вычисление минимального элемента. Вычисление максимального элемента. Замена всех элементов, равных максимальному элементу, минимальным.

4. Конец работы.

III. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Вычисление минимального элемента. Вычисление максимального элемента. Замена всех элементов, стоящих между минимальным и максимальным элементами, на нуль.

4. Конец работы.

IV. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение количества положительных элементов в массиве. Замена всех четных элементов массива на их индексы.

4. Конец работы.

V. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Нахождение минимального элемента. Определение количества элементов, имеющих минимальное значение. Удаление всех минимальных элементов из массива.

4. Конец работы.

VI. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение первого отрицательного элемента в массиве. Вычисление суммы отрицательных элементов. Замена всех элементов, стоящих перед первым отрицательным элементом, на сумму отрицательных элементов.

4. Конец работы.

VII. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение минимального элемента среди элементов, имеющих четный индекс. Определение минимального элемента среди элементов, имеющих нечетный индекс. Замена всех элементов, стоящих между этими минимальными элементами, на их индексы.

4. Конец работы.

VIII. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение минимального элемента среди положительных элементов массива. Замена всех отрицательных элементов массива на этот минимальный положительный элемент.

4. Конец работы.

IX. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Нахождение второго по порядку минимального элемента. Замена всех элементов, кратных трем, на этот элемент.

4. Конец работы.

X. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Нахождение минимального элемента. Вставка после минимального элемента его индекса.

4. Конец работы.

XI. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение максимального элемента. Определение количества элементов, имеющих максимальное значение. Удаление всех максимальных элементов из массива

4. Конец работы.

XII. Создать программу, обеспечивающую выполнение следующих пунктов меню:

1. Ввод массива целых чисел.

2. Вывод массива в строку.

3. Определение количества элементов, стоящих после последнего отрицательного элемента в массиве. Вставка после этого элемента найденного количества.

4. Конец работы.

Лабораторная работа 13. Формирование массива

Теория

Существует целый ряд задач, в которых требуется на основе одного или нескольких исходных массивов получить новый массив. В общем случае для этого следует использовать и новый индекс. В новом массиве необходимо зарезервировать количество элементов с учетом количества элементов в исходном массиве.

Примеры

Пример 1

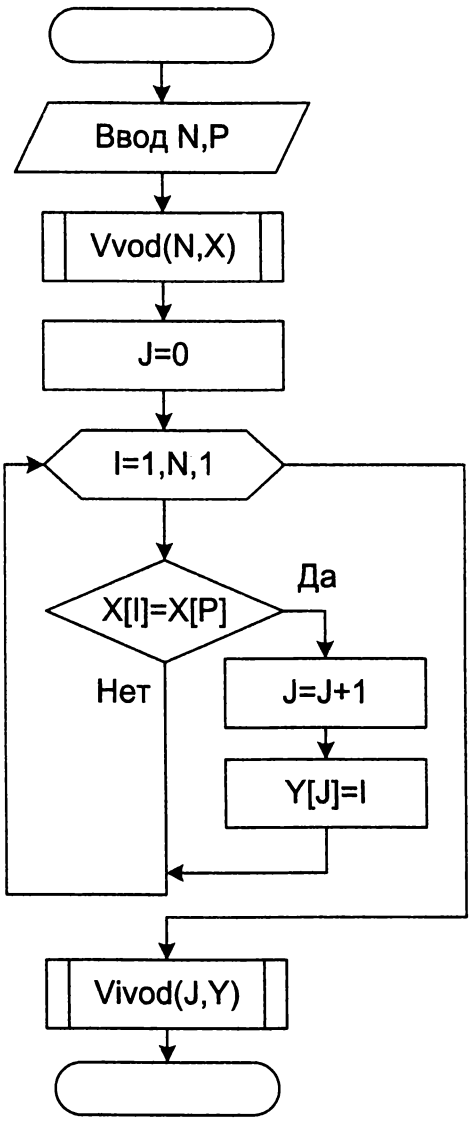
Сформировать новый массив из индексов элементов массива *X*, равных элементу с номером P.

Исходные данные: целочисленный массив *X* из *N* элементов, *P* – номер элемента.

Результат: Целочисленный массив *Y* из *J* элементов.

При описании массива *Y* следует зарезервировать столько же элементов, сколько их в массиве *X*, так как удовлетворять данному условию могут все элементы массива *X*.

Тестовый пример: при *N* = 10, *P* = 1 и массиве *X*: 4, 6, 8, 4, 6, 2, 8, 4, 7, 4 массив *Y*: 1, 4, 8, 10.



Пример 2

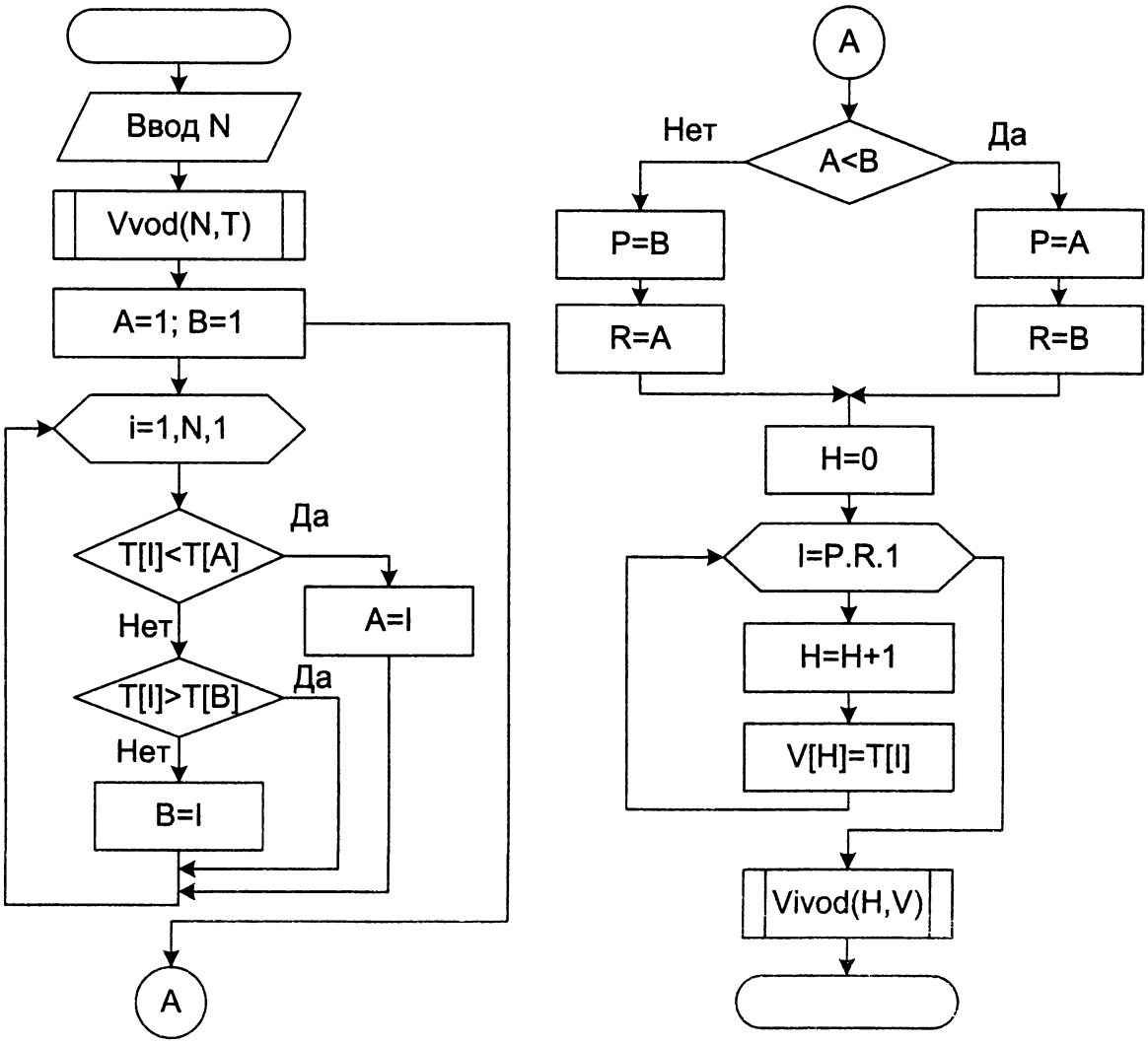
Получить новый массив из элементов данного массива, расположенных между экстремальными элементами включительно.

Дано: целочисленный массив T из N элементов.

Результат: целочисленный массив V из H элементов.

Промежуточные значения: A – номер минимального элемента; B – номер максимального элемента; P – номер левого экстремального элемента; R – номер правого экстремального элемента; H – текущий индекс элементов массива V.

Тестовый пример: при N = 10 и массиве T: 3, 6, 9, 3, 7, 4, 2, 8, 5, 6 массив H: 9, 3, 7, 4, 2.



Контрольные вопросы

1. Если требуется получить новый массив из элементов исходного, сколько элементов нужно для него зарезервировать?

2. В каких ситуациях при получении нового массива не надо использовать новый индекс?

3. Для чего в примере 2 используются переменные *P* и *R*?

4. Если требуется получить новый массив из двух исходных массивов, сколько элементов нужно для него зарезервировать?

Задания для лабораторной работы

1. Дан массив целых чисел. Получить новый массив из данного, удалив все нулевые элементы.

2. Дан массив целых чисел, каждое из которых отлично от нуля. Если в массиве отрицательные и положительные члены чередуются (+, –, +, –, … или –, +, –, +, …), то нужно получить новый массив, совпадающий с данным. Иначе получить новый массив из отрицательных элементов данного массива, сохранив порядок их следования.

3. Дан массив действительных чисел. Получить новый массив, выбросив из исходного все члены, равные максимальному и минимальному элементам данного массива.

4. Дан массив целых чисел. Если в данном массиве ни одно четное число не расположено после нечетного, то получить новый массив, содержащий все отрицательные члены данного массива, иначе – содержащий все положительные. Порядок следования чисел в обоих случаях заменить на обратный.

5. Дан массив целых чисел, в котором могут встречаться повторяющиеся элементы. Получить новый массив, содержащий повторяющиеся элементы данного массива по одному разу.

6. Дан массив действительных чисел. Получить новый массив из элементов данного массива, значение которых больше среднего значения. Элементы в новом массиве должны располагаться в обратном порядке.

7. Дан массив действительных чисел. Получить новый массив из данного без элементов, расположенных до максимального и после минимального элементов этого массива.

8. Дан массив целых чисел. Получить новый массив из положительных четных элементов первоначального массива.

9. Дан массив действительных чисел. Получить новый массив из элементов данного массива, не меньших первого элемента. Элементы в полученном массиве расположить в обратном порядке.

10. Дан массив действительных чисел, содержащий не менее двух нулей. Получить новый массив из элементов данного массива, расположенных между первым и последним нулевыми элементами.

11. Дан массив целых чисел. Получить новый массив из первоначального, отбросив все четные элементы в этом массиве.

12. Дан массив целых чисел. Получить новый массив из элементов данного, следуя правилу: если в первоначальном массиве встречается элемент, равный нулю, переписать в новый массив два элемента – предшествующий нулю и следующий за ним.

13. Дан массив целых чисел. Получить новый массив из положительных нечетных элементов первоначального массива.

Лабораторная работа 14. Двумерный массив

Теория

Данные из прямоугольных таблиц хранятся в двумерных массивах, или матрицах. Каждый элемент двумерного массива характеризуется двумя индексами: первый индекс – это номер строки, в которой находится данный элемент, второй индекс – номер столбца.

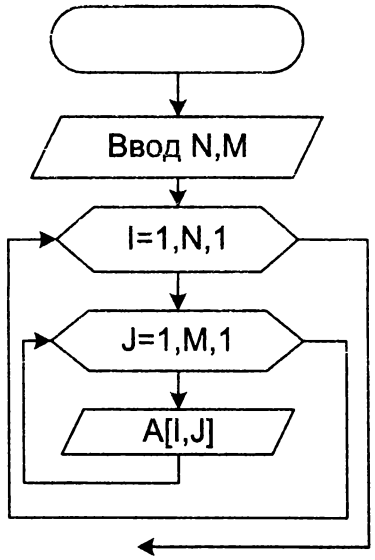
Описание двумерных массивов имеет следующий вид: Var *ИмяМассива*: array [*ТипИндексаСтрок, ТипИндексаСтолбцов*] of *ТипЭлементов;*

Например, var A: array [1..10,1..10] of real;

Двумерный массив часто определяется как массив, элементами которого являются одномерные массивы. Поэтому правила выбора типа индекса у них одинаковые.

Для работы с элементами двумерного массива требуется наличие двух вложенных циклов: внешний цикл перебирает строки, а внутренний – столбцы в этих строках.

Блок-схема и программа ввода двумерного массива (на примере массива A) выглядят следующим образом:




```
program Massiv2;
 var N,M,I,J:integer;
     A: array [1..10,1..10] of integer;
 begin
   write("N,M=');readln(N,M);
   For I:=1 to N do
      For J:=1 to M do
         begin
           write('A[',I,',
',J,']=');readln(A[I,J]);
         end;
    . . .
```

Отметим, что блок-схемы вывода и ввода двумерного массива совпадают.

Рассмотрим процедуру ввода двумерного массива A, у которого количество строк *N*, количество столбцов – *M*. Для использования массива в качестве параметров процедуры необходимо описать тип двумерного массива:

```
Type
mas1=array [1..10, 1..10] of real;
procedure Vvod2(N, M integer, var A: mas2);
var I, J: integer;
begin
For I:=1 to N do
For J:=1 to M do
read (A [I, J];
end;
```

При таком вводе можно вводить данные как таблицу, используя оператора *read*.

Процедура вывода двумерного массива A по строкам выглядит следующим образом:

```
procedure Vivod2(N, M integer, A: mas2);
var I, J: integer;
begin
For I:=1 to N do
begin
```

```
For J:=1 to M do
write (A [I, J]:6:1);
writeln;
end;
end;
```

В двумерном массиве, одновременно являющимся и матрицей, выделяют элементы главной и побочной диагоналей. Элементы главной диагонали удовлетворяют условию $I = J$. Элементы побочной диагонали – условию $I + J = M + 1$.

Примеры

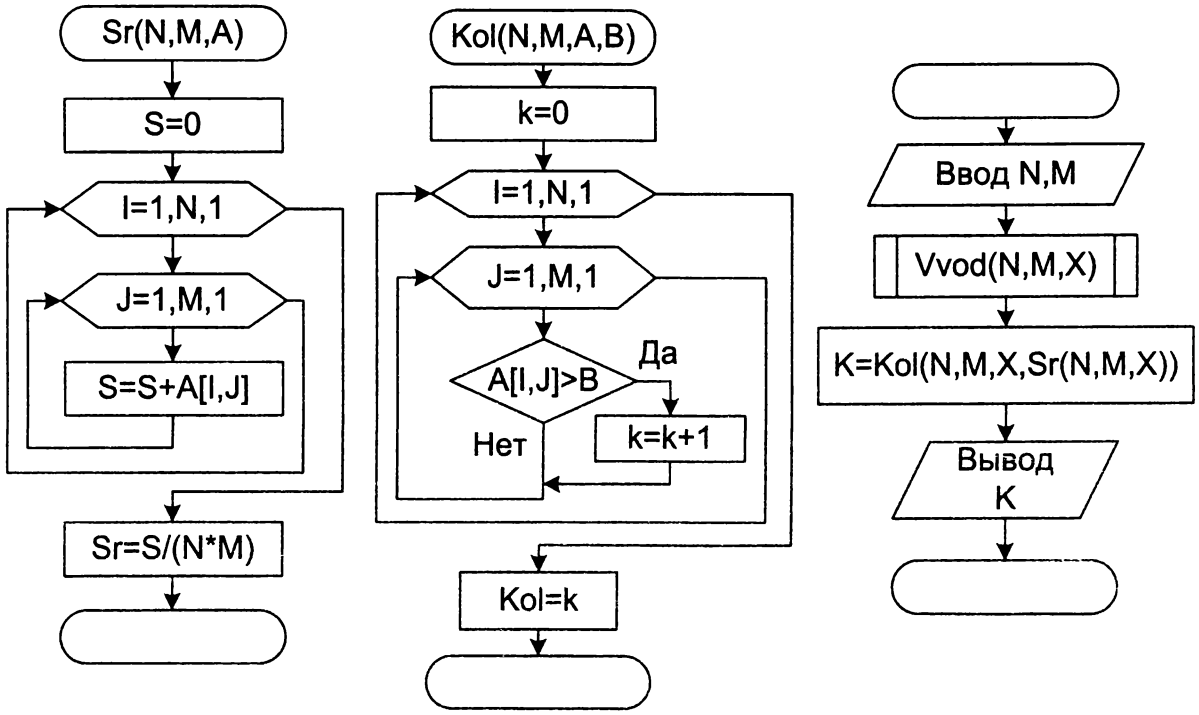
Пример 1

Дан целочисленный двумерный массив. Определить, сколько в этом массиве элементов, больших среднего арифметического.

Исходные данные: целочисленный массив X, количество строк – N, количество столбцов – M.

Результат: K – количество элементов, больших среднего арифметического значения.

Среднее арифметическое значение и количество элементов, больших, чем оно, вычисляется в функциях.

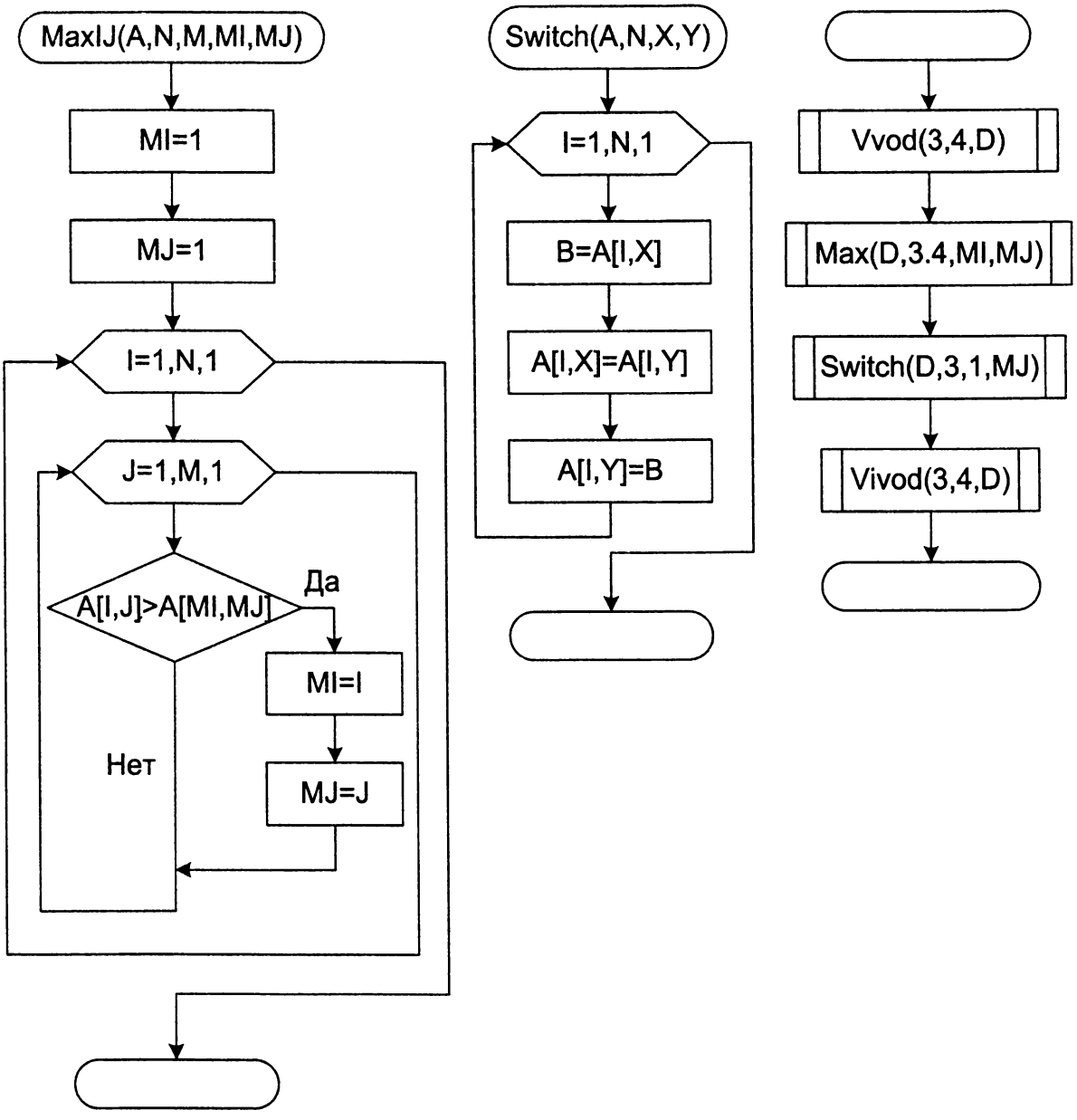


Пример 2

Дан двумерный массив вещественных чисел размером 3 на 4. Поменять местами столбец, содержащий максимальный элемент, с первым столбцом.

Исходные данные: двумерный массив *D* из 3 строк и 4 столбцов.

Результат: преобразованный двумерный массив *D*.

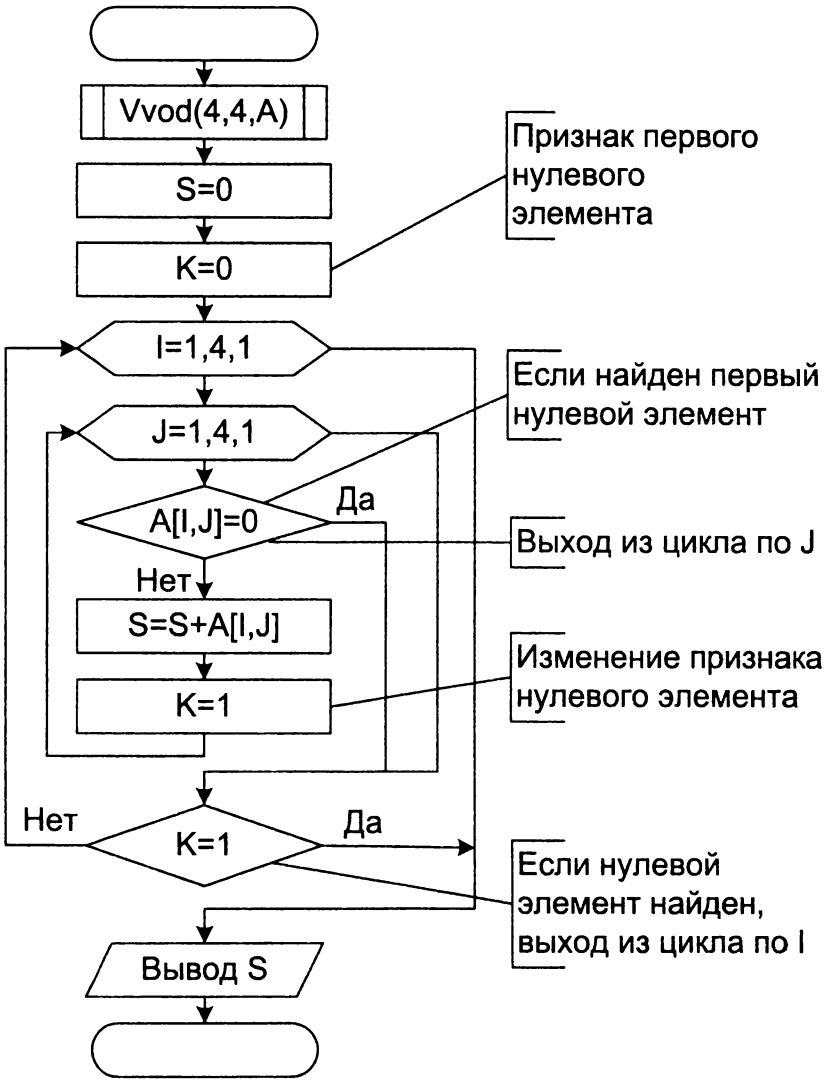


Пример 3

Дан двумерный массив целых чисел. Суммировать элементы, стоящие до первого нуля. Просмотр выполнять по строкам.

Исходные данные: двумерный массив *A* размером 4 на 4.

Результат: S – сумма элементов, предшествующих первому нулевому элементу в массиве.



Контрольные вопросы

1. Сколько элементов в массиве размером 4 на 5?
2. Какое условие выполняется для элементов ниже главной диагонали?
3. Можно ли в примере 2 вместо процедуры *Max* использовать функцию?
4. Укажите параметры-переменные и параметры-значения в процедуре *Switch* из примера 2.
5. Можно ли в примере 3 обойтись без признака нулевого элемента *K*?

Задания для лабораторной работы

1. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Поменять местами строку, содержащую минимальный элемент, и последнюю строку.

2. Дан двумерный массив вещественных чисел из 3 столбцов и 4 строк. Определить, сколько в каждом столбце отрицательных элементов.

3. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Найти сумму максимальных элементов каждой строки.

4. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Определить, сколько нулей находится выше главной диагонали.

5. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти максимальный элемент в массиве и заменить его нулем.

6. Дан двумерный массив вещественных чисел из 3 столбцов и 4 строк. Все элементы в массиве, стоящие после минимального, заменить первым элементом. Замену выполнять по строкам.

7. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Первый элемент в каждом столбце заменить на среднее арифметическое от всех элементов массива.

8. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Все элементы главной диагонали заменить на сумму элементов в данном массиве, имеющих четное значение.

9. Дан двумерный массив вещественных чисел из 4 столбцов и 3 строк. Поменять местами первый столбец и столбец, в котором находится максимальный элемент массива. Вывести массив по строкам до и после перестановки.

10. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Найти минимальный элемент в первой половине массива (просмотр вести по строкам) и во второй. Поменять местами эти минимальные элементы.

11. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти сумму элементов в этом массиве, стоящих после первого нуля.

12. Дан двумерный массив целых чисел из 4 столбцов и 3 строк. Найти максимальный элемент среди отрицательных элементов этого массива. Вывести массив по строкам.

13. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Найти минимальный элемент и обнулить все элементы правее и ниже него.

14. Дан двумерный массив целых чисел из 4 столбцов и 4 строк. Поменять знак у всех элементов, расположенных ниже побочной диагонали этого массива.

Лабораторная работа 15. Записи

Теория

Запись – это структурированный тип данных, состоящий из определенного числа компонентов, которые называются полями записи. Они могут иметь разный тип.

Описание переменной типа *запись* начинается ключевым словом *record*. За ним следует описание списка полей с указанием их типов. Заканчивается описание служебным словом *end*. Записи лучше описывать в разделе описания типов.

```
type
abc=record
a, b: string;
c, d: real;
end;
var sk1: abc;
```

Для того чтобы обратиться к полю записи, необходимо указать имя переменной и – через точку – имя поля: sk1.c=sk1.d+17;

Или с помощью оператора *with*: With *ПеременнаяЗапись do* операторы;

Например:

```
with sk1 do
begin
.c=c+1;
.d=d+0.1;
end;
```

Обработка отдельных полей определенных типов осуществляется так же, как и переменных данного типа.

Записи часто используют при работе с таблицами, где каждая запись – это строка таблицы. Следовательно, для обработки всей таблицы необходимо использовать массивы записей. Например:

```
type
ved=record
fio: string;
date: integer;
zarplata: real;
end;
var a: array [1..20] of ved;
```

Для того чтобы обратиться к полю fio в пятой строке массива *a*, достаточно указать соответствующий элемент массива *a* – *a*[5].fio.

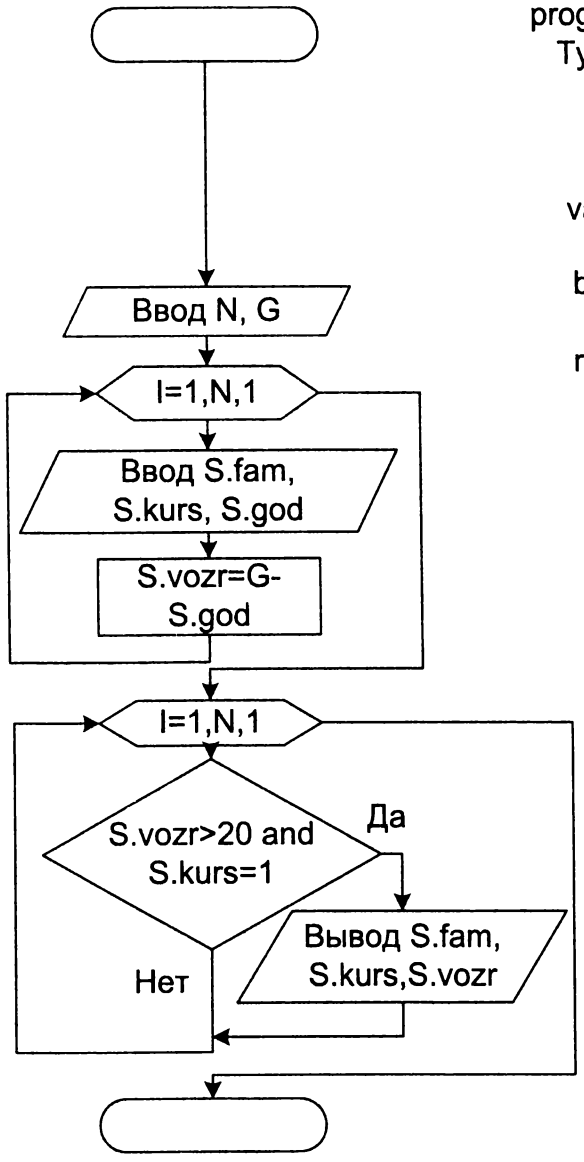
Если переменные записи имеют одинаковый тип, их можно присваивать друг другу.

Примеры

Дан список студентов с указанием фамилии, года рождения и курса. Вычислить возраст студентов. Вывести фамилии студентов первого курса старше 20 лет.

Исходные данные: *S* – массив записей с полями fam – строковый тип, kurs, god и vozr – целый тип, *G* – текущий год.

Результат: вывод элементов массива, удовлетворяющих заданному условию.




```
program Zapisi;
  Type Stud=record
          fam: string[20];
          god, kurs, vozr:integer;
        end;
  var N, G, I: integer;
      S:array[1..20] of Stud;
  begin
       write('Введите N и текущий год');
  readln(N,G);
       for I:=1 to N do
         with S[I] do
            begin
               write('fam='); readln(fam);
               write('kurs='); readln(kurs);
               write('god='); readln(god);
               vozr:=G-god;
            end;
      for I:=1 to N do
        with S[I] do
           begin
             if (vozr>20) and (kurs=1) then
                writeln( fam:20, kurs:3, vost:4);
           end;


end.
```

Контрольные вопросы

1. Для чего используется оператор *with*?

2. В каких случаях для переменных типа запись можно выполнять операцию C:=D?

3. Может ли запись быть элементом массива?

4. Какого типа могут быть поля записи?

Задания для лабораторной работы

1. Соревнования по фигурному катанию. Ввести данные, содержащие фамилию спортсмена, страну и оценки за обязательные выступления и произвольную программу. Вычислить суммарную оценку. Определить победителя.

2. Дан список товаров, содержащий следующие данные: название товара, стоимость покупки, стоимость продажи, количество проданного товара. Найти прибыль по каждому товару. Вывести список товаров с прибылью выше средней.

3. Соревнования по прыжкам с трамплина. Список содержит фамилии участников и их результаты по трем попыткам. Вычислить суммарный результат. Найти самый дальний прыжок.

4. Дан список путевок, содержащий следующие данные: название курорта, страна, стоимость проживания и проезда. Вывести суммарную цену путевки.

5. Дан список телевизоров, содержащий следующие данные: марка, стоимость, магазины, в которых они продаются, и предоставляемая скидка. Вычислить стоимость телевизоров с учетом скидки. Вывести самый дешевый, с учетом скидки, телевизор.

6. Дан список коробок конфет, содержащий следующие данные: название коробки, цена одной конфеты в коробке, количество конфет в коробке. Вычислить суммарную стоимость каждой коробки. Найти самую дешевую из них.

7. Дан список студентов, содержащий следующие данные: фамилия студента, оценки по физике, математике и информатике. Получить данные для поля «стипендия» и вывести фамилии студентов, получивших стипендию.

8. Дан список книг, содержащий следующие данные: название, год издания, издательство, первоначальная стоимость. Вычислить текущую стоимость книг, учитывая, что каждый год она уменьшается на 10 %. Вывести список книг, подлежащих списанию (тех, чья текущая цена составляет меньше 10 % от первоначальной стоимости).

9. Дан список, содержащий следующие данные о студентах: фамилия, курс, факультет, год рождения. Вычислить возраст студентов. Вывести список студентов первого курса.

10. Дан список сотрудников, содержащий следующие сведения: фамилия сотрудника, отдел, в котором он работает, оклад и премия. Вычислить суммарную зарплату каждого сотрудника. Вывести список сотрудников с зарплатой ниже средней.

11. Дан список дисков, который содержит следующие данные: название диска, количество содержащихся на нем фильмов, их продолжительность и стоимость диска. Вычислить среднюю продолжительность каждого фильма. Вывести названия дисков, на которых записан один фильм.

12. Дан список газет, содержащий следующие данные: название газеты, периодичность выхода, цена одного экземпляра. Вычислить стоимость подписки на год. Вывести список газет, выходящих раз в неделю.

Лабораторная работа 16. Сортировка

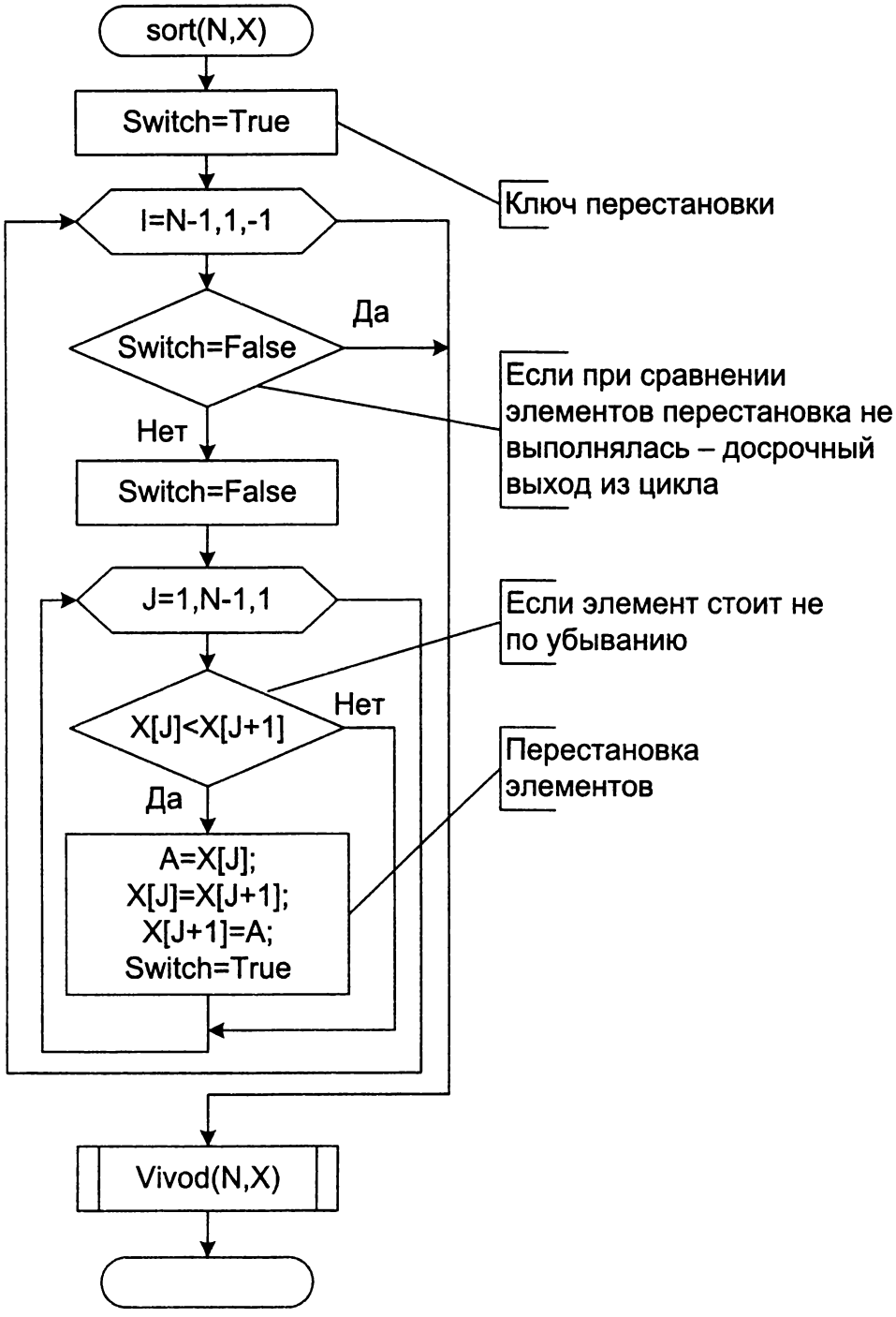
Теория

Сортировка – это процесс упорядочивания набора данных одного типа по возрастанию или убыванию значения какого-либо признака. Чаще всего используются сортировка массивов, сортировка строк и сортировка записей.

При сортировке элементы массива меняются местами так, что их значения оказываются упорядоченными или по возрастанию, или по убыванию.

Существует множество алгоритмов сортировки, отличающихся друг от друга скоростью работы. Мы рассмотрим один из самых простых алгоритмов: сортировка методом «пузырька», который основан на том, что в процессе исполнения алгоритма более «легкие» элементы «всплывают». При использовании этого метода сравниваются соседние элементы, а потом, если нужно, переставляются так, что меньший элемент оказывается левее

(если предполагается сортировка по возрастанию) или правее (если требуется сортировка по убыванию). Вот как выглядит блок-схема алгоритма сортировки в виде процедуры *Sort*, сортирующей массив *X* из *N* элементов:



Примеры

Пример 1

Дан двумерный массив целых чисел. Отсортировать элементы выше главной диагонали по возрастанию.

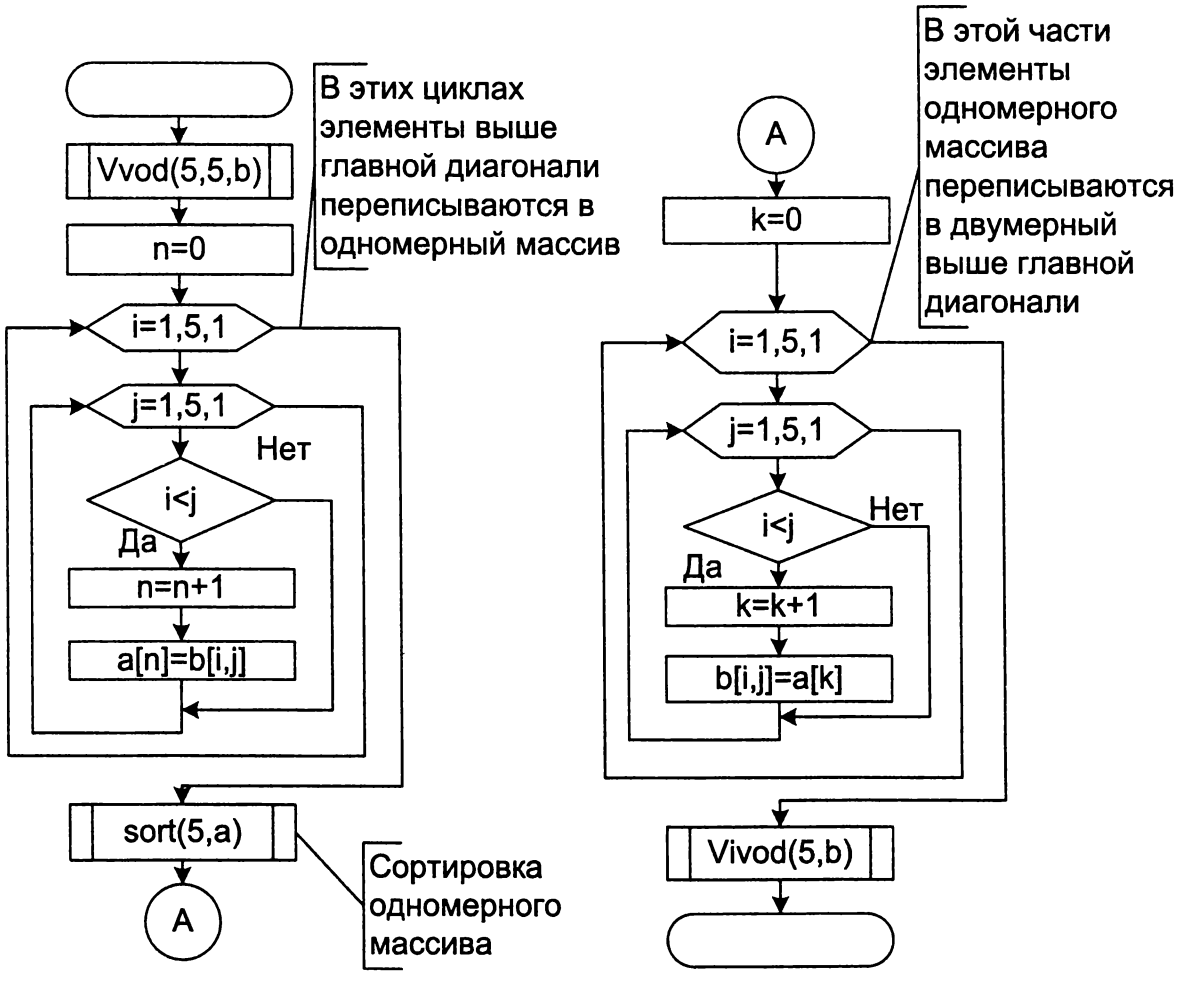
Исходные данные: двумерный массив целых чисел *B* из 5 строк и 5 столбцов.

Результат: двумерный массив B с отсортированными элементами выше главной диагонали.

Сначала в одномерный массив переписываются элементы выше главной диагонали. Далее эти элементы сортируются по возрастанию методом «пузырька». Затем они переписываются назад в двумерный массив выше главной диагонали.

Тестовый пример:

Исходный массив					Результат				
2	4	6	8	4	2	**2**	**3**	**4**	**4**
3	2	5	4	7	3	2	**4**	**4**	**5**
1	4	6	3	4	1	4	6	**6**	**7**
8	4	9	9	2	8	4	9	9	**8**
3	5	7	4	6	3	5	7	4	6



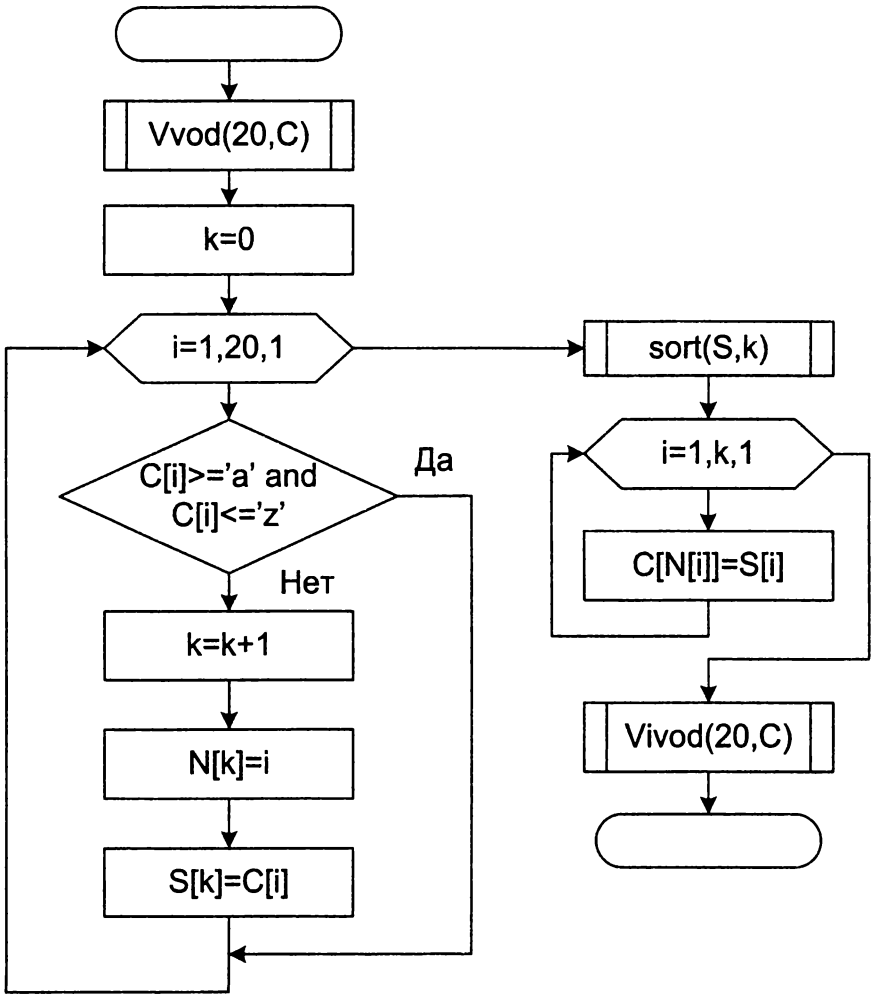
Пример 2

Дан массив символов. Отсортировать в этом массиве символы букв в порядке возрастания. Остальные символы оставить на месте.

Исходные данные: массив символов *C* из 20 элементов.

Результаты: массив *C* с отсортированными символами букв.

При решении использовались два вспомогательных массива: *S* – массив символов букв и *N* – массив, в который записывалось положение символов букв. Затем массив *S* сортировался, после чего с помощью массива *N* буквы возвращались в первоначальный массив, но уже в отсортированном виде.



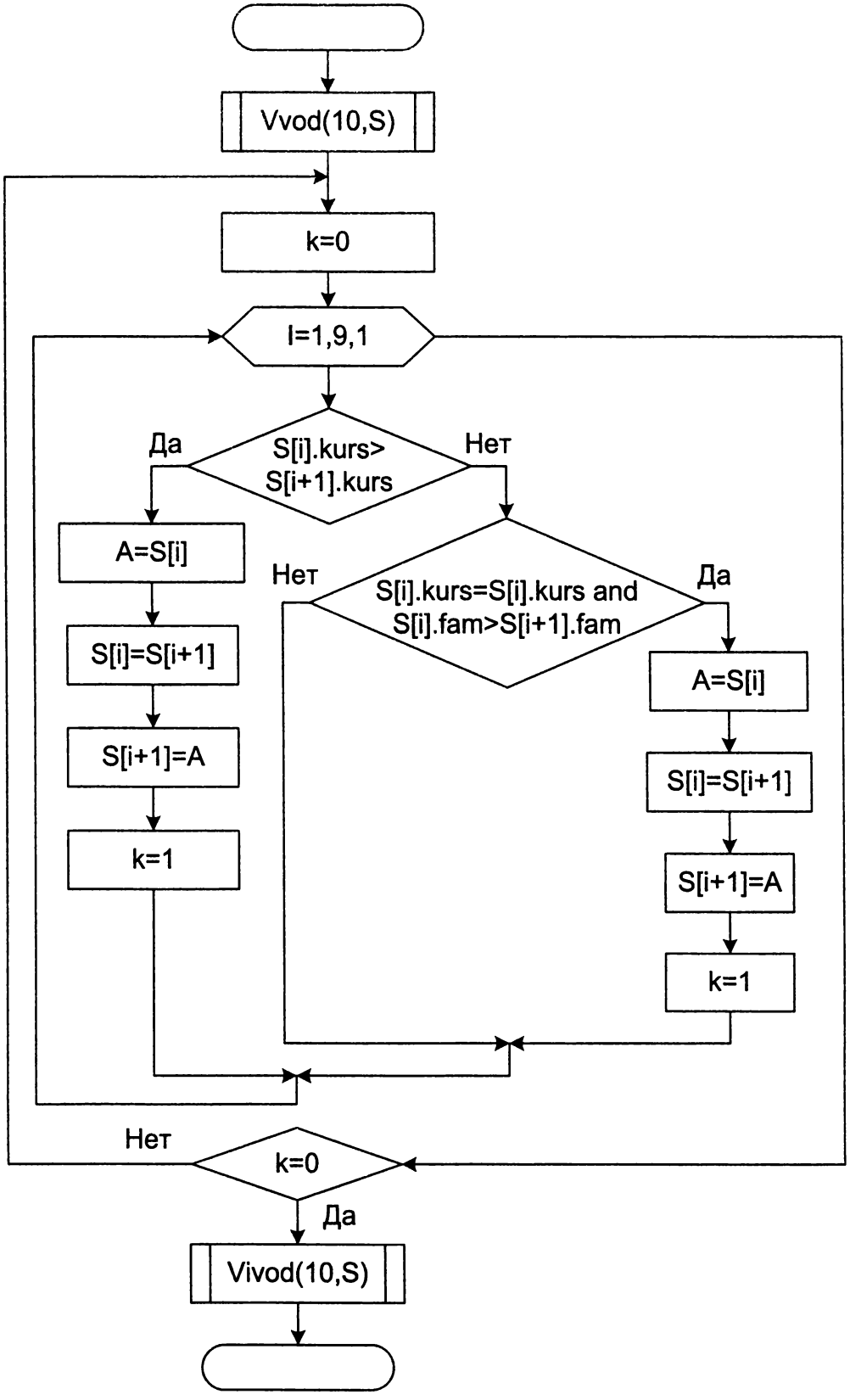
Пример 3

Дан массив записей, содержащих поля «фамилия» и «курс». Отсортировать список по курсам, а внутри курса – по фамилиям.

Исходные данные: 10 элементов массива записей *S* с полями fam – строкового типа и kurs – целого типа.

Результат: отсортированный массив *S*.

Для проверки того, была ли выполнена перестановка при сравнении элементов массива, при сортировке используется переменная *k*.



Контрольные вопросы

1. Сравните алгоритм, приведенный в теоретической части, и алгоритм сортировки и процедуры в примере 1. Чем они отличаются?

2. Как изменить сортировку по убыванию на сортировку по возрастанию?

3. В массиве слов выполняется сортировка. Будет ли она выполняться по вторым и третьим буквам при совпадении первых букв?

Задания для лабораторной работы

1. Отсортировать массив записей, содержащих информацию о турнире по футболу (название команды, количество побед, поражений и ничьих), по убыванию очков.

2. Отсортировать элементы одномерного массива между минимумом и максимумом в порядке возрастания.

3. Дан список студентов, содержащий фамилии студентов и их оценки по физике, математике и информатике. Определить стипендию для каждого студента (стипендия может быть повышенной – все «пятерки», обычной и стипендии может не быть). Вывести список студентов, получающих стипендию, отсортированный сначала в порядке убывания стипендии, а потом по алфавиту.

4. Отсортировать первую половину одномерного массива по возрастанию, а вторую – по убыванию.

5. Дан список путевок, содержащий следующие данные: название курорта, страна, стоимость проживания и проезда к месту отдыха. Получить список, в котором сначала российские путевки отсортированы в порядке возрастания стоимости, а потом зарубежные – в порядке убывания стоимости.

6. В массиве слов отсортировать слова, стоящие на четных местах в порядке возрастания, а на нечетных местах – в порядке убывания.

7. Дан массив записей, содержащих информацию о пациентах: фамилия, пол, возраст. Отсортировать массив сначала по полу, а потом по возрасту в порядке возрастания.

8. Сгенерировать 20 элементов одномерного массива целых чисел в интервале от –100 до 100. Отсортировать все отрицательные элементы в порядке возрастания, а все положительные – в порядке убывания.

9. Дан двумерный массив целых чисел с одинаковым количеством строк и столбцов. Отсортировать элементы главной диагонали по возрастанию.

10. Дан двумерный массив целых чисел. Выполнить сквозную сортировку по убыванию всех элементов этого массива.

11. Дан двумерный массив вещественных чисел. Отсортировать этот массив по строкам.

12. Дан двумерный массив целых чисел с одинаковым количеством строк и столбцов. Отсортировать элементы выше главной диагонали по возрастанию, ниже главной диагонали – по убыванию.

Тема 6. АЛГОРИТМЫ РАБОТЫ С ФАЙЛАМИ

Данные могут храниться во внешних файлах, связь с которыми выполняется через переменные файлового типа. Существуют три ***типа файлов:***

- текстовые;
- типизированные;
- нетипизированные.

Схема работы с файлом любого типа следующая:

1. Определение переменной файлового типа.

2. Установление связи между переменной файлового типа и физическим файлом.

Это действие выполняет процедура *Assign* (*файловая переменная, имя файла*). В качестве имени файла может выступать строковая константа или переменная строкового типа. Если файл находится в текущем каталоге, указывается только имя файла, если в другом – указывается полное имя файла.

3. Открытие или создание файла. При этом указывается, для чего открывается файл: для чтения данных или для записи.

Создание файла связано с процедурой *Rewrite* (*файловая переменная*): если файл с таким именем в данной папке уже существует, он удаляется и создается новый файл с тем же именем. Открытие уже существующего файла связано с процедурой *Reset* (*файловая переменная*).

4. Непосредственная запись в файл или чтение данных из файла.

5. Закрытие файловой переменной в конце работы.

Файл закрывается в конце работы программы процедурой *Close* (*файловая переменная*).

Существует еще ряд процедур и функций, общих для всех типов файлов:

- функция *IOResult* (*файловая переменная*) проверяет существование данного файла. Если файл существует, значение функции равно 0, иное значение свидетельствует о том, что в указанной папке файла нет;
- процедура *Rename* (*файловая переменная, имя файла*) позволяет переименовать файл. Но использовать эту процедуру можно только когда файл будет закрыт;
- процедура *Erase* (*файловая переменная*) удаляет файл. В этом случае файл также должен быть закрыт.

Для того чтобы работать с данными в файле, необходимо просматривать все его записи. Логическая функция *Eof* (*файловая переменная*) равна функции *True*, если достигнут конец файла. Поэтому она активно используется для проверки этого факта.

Лабораторная работа 17. Текстовые файлы

Теория

Текстовые файлы связываются с файловыми переменными, принадлежащими стандартному типу *TextFile* и предназначены для хранения текстовой информации. Именно в такого типа файлах хранятся, например, исходные тексты программ. Компоненты (записи) текстового файла могут иметь переменную длину, что существенно влияет на характер работы с ними.

Текстовый файл трактуется в *Pascal* как совокупность строк переменной длины. Доступ к каждой строке возможен лишь последовательно, начиная с первой. При создании текстового файла в конце каждой строки ставится специальный признак *EOLN (End Of LiNe* – конец строки), а в конце всего файла – признак *EOF (End Of File* – конец файла). Эти признаки можно протестировать одноименными логическими функциями, речь о которых пойдет ниже. При формировании текстовых файлов используются следующие системные соглашения:

- *EOLN* – последовательность кодов #13 (*CR*) и #10 (*LF*);
- *EOF* – код#26.

Для создания текстового файла используется процедура *Rewrite*, а для дополнения файла новыми данными – процедура *Append*. При этом данные записывают в конец файла.

Для доступа к записям применяются процедуры *Read, ReadLn, Write, WriteLn.* Параметры этих процедур такие же, как и при вводе и выводе на экран, но первым параметром в любой из перечисленных процедур должна стоять файловая переменная.

Процедура *Read* предназначена для последовательного чтения из текстового файла символьных представлений переменных V*i*. При чтении переменных типа *Char* выполняется чтение одного символа и присваивание считанного значения переменной. Если перед выполнением чтения указатель файла достиг конца очередной строки, то результатом чтения будет символ *CR* (код #13), а если достигнут конец файла, то символ *EOF* (код

#26). Процедуру *Read* не рекомендуется использовать для ввода переменных типа *String,* так как она не способна «перепрыгнуть» через разделитель строк *EOLN* и читает только первую строку текстового файла. Для ввода последовательности строк нужно использовать процедуру *ReadLn.*

Процедура *Read* прекрасно приспособлена к вводу чисел. При обращении к ней за вводом очередного целого или вещественного числа процедура «перескакивает» маркеры конца строк, т. е. фактически весь файл рассматривается ею как одна длинная строка, содержащая текстовые представления чисел. В сочетании с проверкой конца файла функцией *EOF* процедура *Read* позволяет организовать простой ввод массивов данных.

Процедура *ReadLn* идентична процедуре *Read*, за исключением того, что после считывания последней переменной оставшаяся часть строки до маркера *EOLN* пропускается, поэтому следующее обращение к *ReadLn* начинается с первого символа новой строки.

Процедура *Write* обеспечивает вывод в текстовый файл группы переменных. Любой элемент списка вывода может иметь форму

OutExpr [: MinWidth [: DecPlaces]]

Здесь *OutExpr* – выводимое выражение; *MinWidth, DecPlaces* – выражения типа *Integer* (квадратные скобки означают возможность отсутствия заключенных в них параметров). Параметр *MinWidth*, если он присутствует, указывает минимальную ширину поля, в которое будет записываться символьное представление значения *OutExpr*. Если символьное представление имеет меньшую длину, чем *MinWidth*, оно будет дополнено слева пробелами, если большую, то *MinWidth* игнорируется и в файл помещается необходимое число символов. Параметр *DecPlaces* задает количество десятичных знаков в дробной части вещественного числа. Он может использоваться только совместно с *Min Width* и только по отношению к выводимому выражению одного из вещественных типов.

Если ширина поля вывода не указана, соответствующий параметр выводится вслед за предыдущим без какого-либо их разделения. Символы и строки передаются выводному файлу без изменений, но снабжаются ведущими пробелами, если задана ширина поля вывода и эта ширина больше требуемой для вывода.

Процедура *WriteLn* (*Write Line* – писать строку) полностью идентична процедуре *Write*, за исключением того, что выводимая последовательность символов автоматически завершается маркером *EOLN*.

Примеры

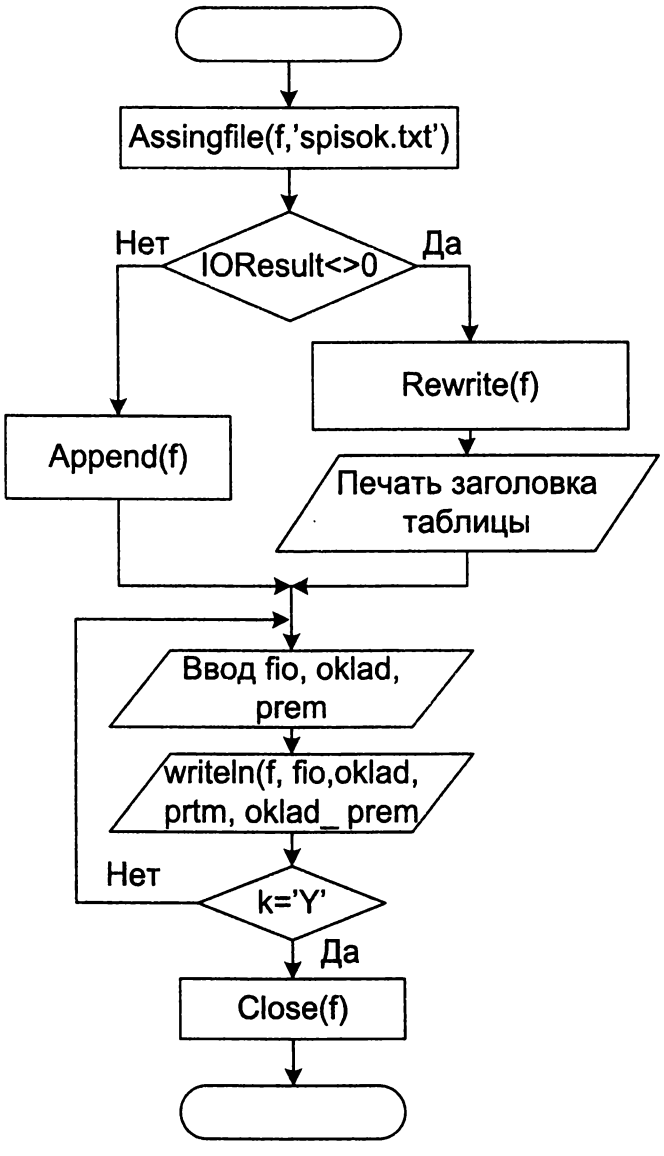
Пример 1

Ввести данные о сотрудниках: фамилия, размер оклада, премии. Сохранить эти данные в текстовом файле в виде таблицы со столбцами: «Фамилия», «Оклад», «Премия» и «Зарплата». Обеспечить возможность дополнения этого файла новыми данными.

Исходные данные: ввод данных в переменные fio – фамилия, oklad – оклад, prem – премия.

Файл 'spisok.txt' может существовать, в этом случае его следует открыть для дополнения. Иначе этот файл нужно создать и ввести первую строку, содержащую заголовок таблицы.

Результат: файл 'spisok.txt' с заполненными данными.

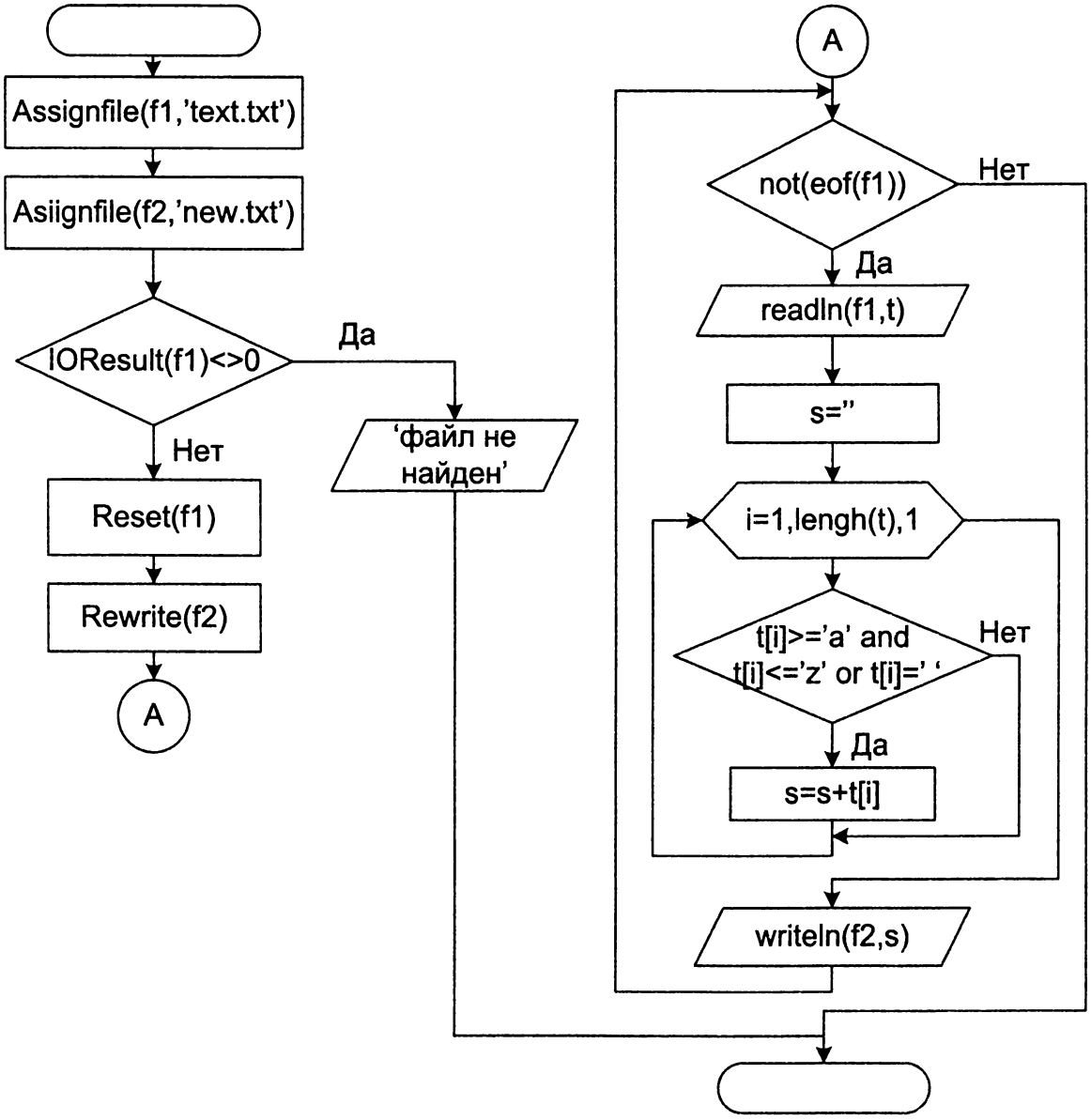


Пример 2

Дан текстовый файл, содержащий слова, разделенные пробелами. Переписать в новый файл только слова, написанные на английском языке.

Исходные данные: файл ‘text.txt’, содержащий первоначальный текст.

Результаты: файл ‘New.txt’ – новый файл.



Пример 3

Дан текстовый файл, содержащий учебный тест. Создать программу теста с указанием процента правильных ответов. Максимальный балл за каждый ответ – 5.

Структура файла:

Вопрос1

ответ1

балл за ответ1

ответ2

балл за ответ2

ответ3

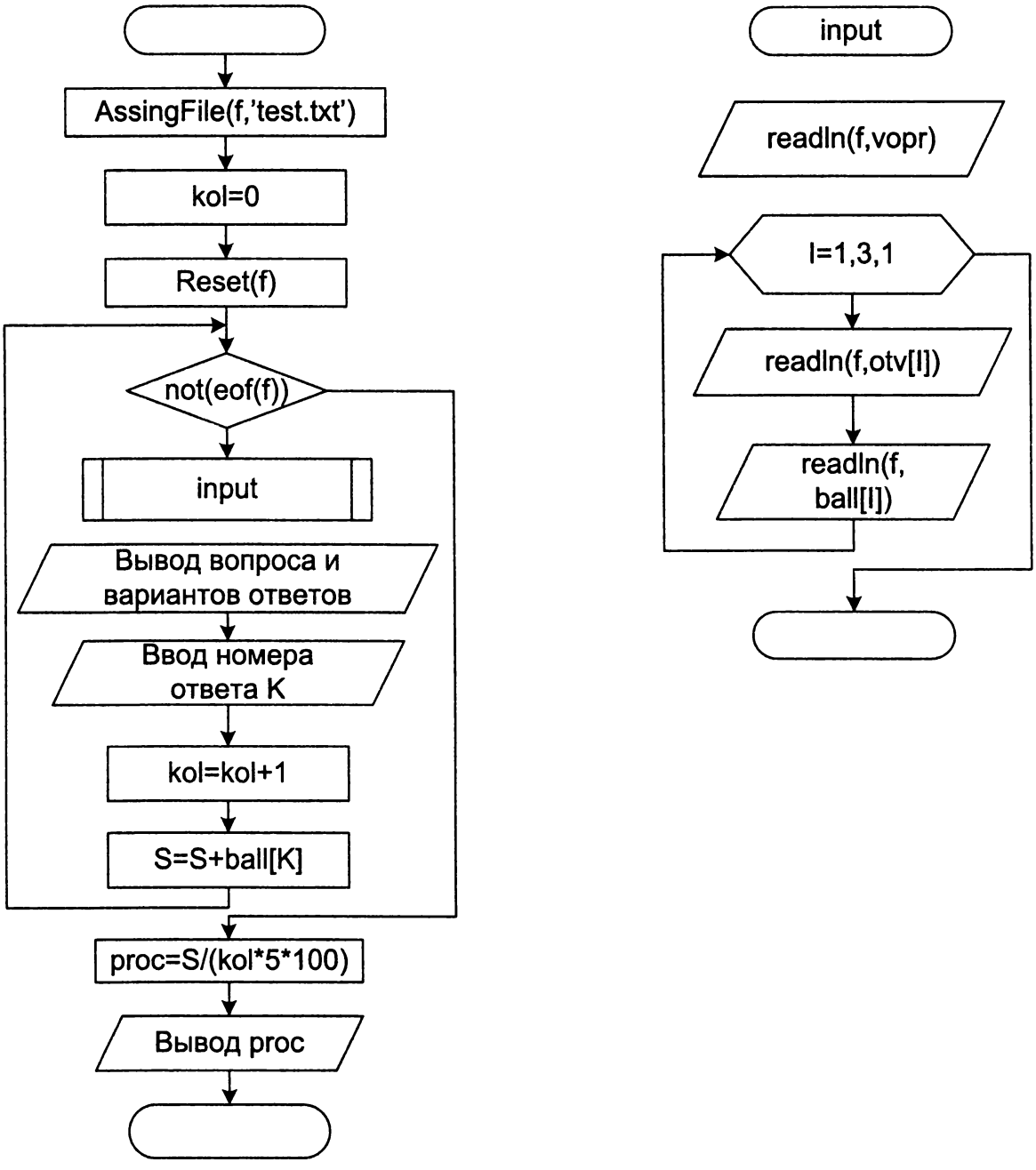
балл за ответ3

Вопрос2

и т. д.

Исходные данные: файл с тестом ‘test.txt’.

Результат: proc – процент правильных ответов.



Контрольные вопросы

1. Какая процедура используется для чтения информации, содержащейся в файле?

2. Что произойдет, если открыть файл через процедуру *Rewrite*?

3. Можно ли считывать числовые данные непосредственно из файла?

4. Чем отличается считывание данных с помощью процедуры *Read* от тех же действий, производимых с помощью процедуры *Readln*?

5. Какая процедура закрывает файл?

6. Для чего используется процедура *IOResult*?

7. Можно ли при открытии текстового файла сразу же прочитать данные из третьей строки?

Задания для лабораторной работы

1. Дан файл, содержащий произвольный текст. Выяснить, чего в нем больше: русских букв или цифр.

2. Дан файл, содержащий текст на русском языке. Подсчитать в нем количество слов, начинающихся и заканчивающихся на одну и ту же букву.

3. Дан файл, содержащий произвольный текст. Проверить, правильно ли в нем расставлены круглые скобки (т. е. находится ли правее каждой открывающейся скобки закрывающаяся и левее закрывающейся – открывающаяся).

4. Дан текстовый файл. Удалить из него все лишние пробелы, оставив между словами не более одного пробела. Результат поместить в новый файл.

5. Дан файл, содержащий текст на русском языке. Определить, сколько раз в нем встречается самое длинное слово.

6. Дан файл, содержащий текст на русском языке. Выбрать из него те символы, которые встречаются в нем только один раз, и именно в том порядке, в котором они встречаются.

7. Дан файл, содержащий текст на русском языке. Составить в алфавитном порядке список всех слов, встречающихся в этом тексте, и вывести его в другом файле.

8. Дан файл, содержащий текст и арифметические выражения вида a&*b*, где & – это знак «+», «–», «*» или «/». Выписать все возможные арифметические выражения и вычислить их значения.

9. Дан файл, содержащий сведения о сотрудниках учреждения по следующему образцу: фамилия имя отчество, фамилия имя отчество, … Записать эти сведения в другой файл по образцу: имя отчество фамилия, имя отчество фамилия, … .

10. Дан текстовый файл, содержащий следующие сведения о товаре и символы: название товара, тире, количество товара, запятая, цена товара. В новом текстовом файле напечатать в два столбика название товара и вырученную за него сумму.

11. Дан текстовый файл, содержащий сведения об учениках в следующем виде: фамилия, класс, оценка по математике, отметка по физике. Создать файл *Otlich.txt*, куда переписать фамилии, класс и отметки отличников, и файл *Dvoech.txt*, содержащий информацию о двоечниках. Данные записать в виде таблицы.

12. Создать тест. Данные теста считать из следующего текстового файла:

Умеете ли вы ухаживать за больным?

1. Буду исполнять все капризы больного – 2.

2. Скажу, что не следовало доводить себя до такого состояния – 0.

3. Буду стараться вместе с ним – 2.

4. Не буду обращать внимания на его страдания – 0.

5. Буду за ним ухаживать – 3.

6. Буду подшучивать над его болезнью – 0.

7. Скажу ему, чтобы не притворялся – 0.

8. Продумаю, действительно ли он нуждается в столь заботливом уходе – 4.

9. Применю соответствующие средства для излечения болезни – 9.

10. Буду его баловать – 1.

11. Немедленно вызову врача – 3.

12. Возложу заботы по уходу за ним на третье лицо – мать, свекровь, соседку – 0.

Баллы приведены для утвердительного ответа.

Ответы: 7 и более баллов – Вы правильно ухаживаете за больным; от 6 до 3 баллов – Вам не достает любви; менее 3 баллов – Вы плохая сиделка.

Лабораторная работа 18. Типизированный файл

Теория

Длина любого компонента типизированного файла строго постоянна, что дает возможность организовать прямой доступ к каждому из них (т. е. доступ к компоненту по его порядковому номеру).

Для работы с типизированными файлами используются процедуры *Rewrite* – для создания файла и записи в него, и процедура *Reset* – для чтения и записи в уже существующий файл. А также определенные подпрограммы (табл. 7).

Таблица 7

Подпрограммы, используемые для работы с типизированными файлами

Подпрограмма	Назначение
Procedure Read (var F, V1, ..., Vn);	Чтение данных из типизированного файла F: *V* – переменные такого же типа, что и компоненты файла
Procedure Write (var F, Pl, ..., Pn);	Запись данных в типизированный файл F: *Pi* – выражения такого же типа, что и компоненты файла
Procedure Seek (var F; N: Longlnt);	Смещение указателя файла F к требуемому компоненту: N – номер компонента файла (первый компонент файла имеет номер 0)
Function FileSize (var F); Longlnt;	Возврат количества компонентов файла. Чтобы переместить указатель в конец типизированного файла, можно написать: *seek (FileVar, FileSize (FileVar))*;
Function FilePos (var F): Longlnt;	Возврат текущей позиции в файле, т. е. номера компонента, который будет обрабатываться следующей операцией ввода-вывода

Перед первым обращением к процедурам ввода-вывода указатель файла стоит в его начале и указывает на первый компонент с номером 0. После каждого чтения или записи указатель сдвигается к следующему компоненту файла. Переменные в списках ввода-вывода должны иметь тот же тип, что и компоненты файла. Если этих переменных в списке несколько, указатель будет смещаться после каждой операции обмена данными между переменными и дисковым файлом.

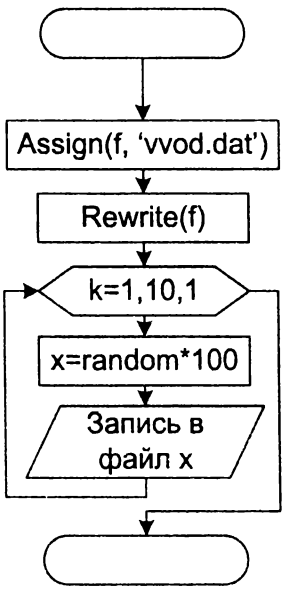
Примеры

Пример 1

Генерировать 10 вещественных чисел от 0 до 100 и записать их в типизированный файл.

Исходные данные: x – переменная вещественного типа, полученная генерацией чисел.

Результат: типизированный файл 'vvod.dat', содержащий вещественные числа.




```
program rnd;
  var f: file of real;
      x: real;
      k: integer;
begin
   Assign(f, 'vvod.dat');
   Rewrite(f);

   For k:=1 to 10 do
begin
        x:=random*100;
        write(f, x);
      end;


end.
```

Пример 2

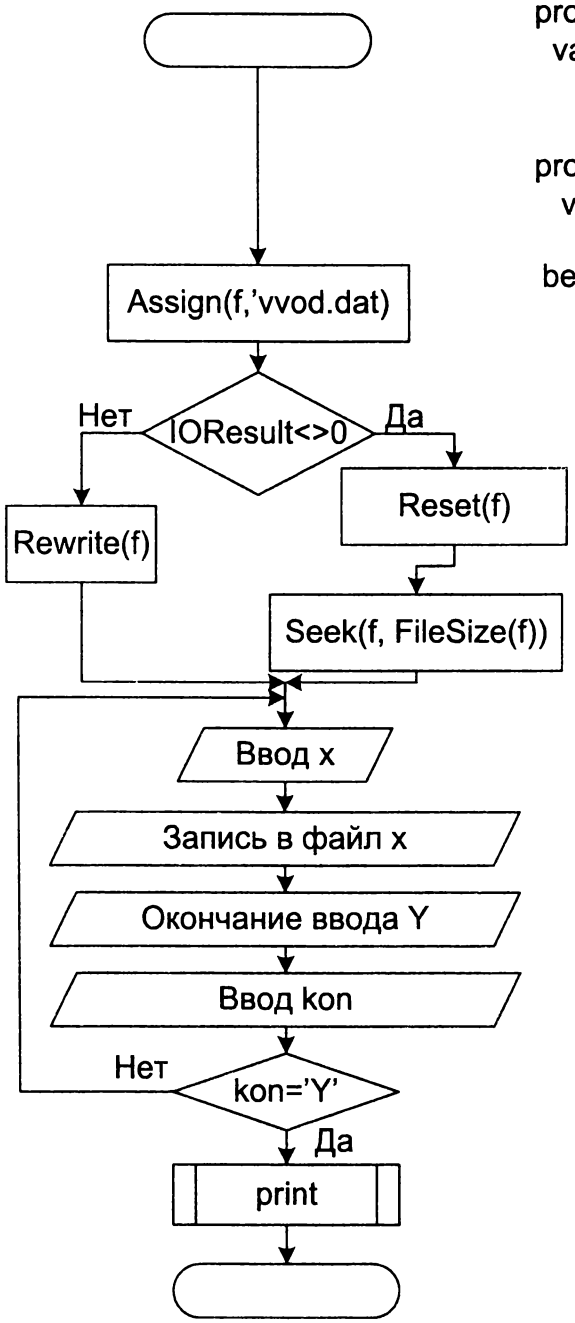
Дополнить типизированный файл, содержащий вещественные числа, новыми данными. Вывести данные из этого типизированного файла на экран. Вывод оформить как процедуру.

Исходные данные: типизированный файл 'vvod.dat', содержащий вещественные числа; x – переменная вещественного типа, в которую данные вводятся. Так как количество данных не указано, их ввод прекращается по запросу.

Результат: созданный или дополненный вещественными числами типизированный файл 'vvod.dat'.

Предполагается, что если файла 'vvod.dat' нет, он должен быть создан.

Следует отметить, что файловая переменная не может быть параметром процедуры. Поэтому она должна быть глобальной переменной.



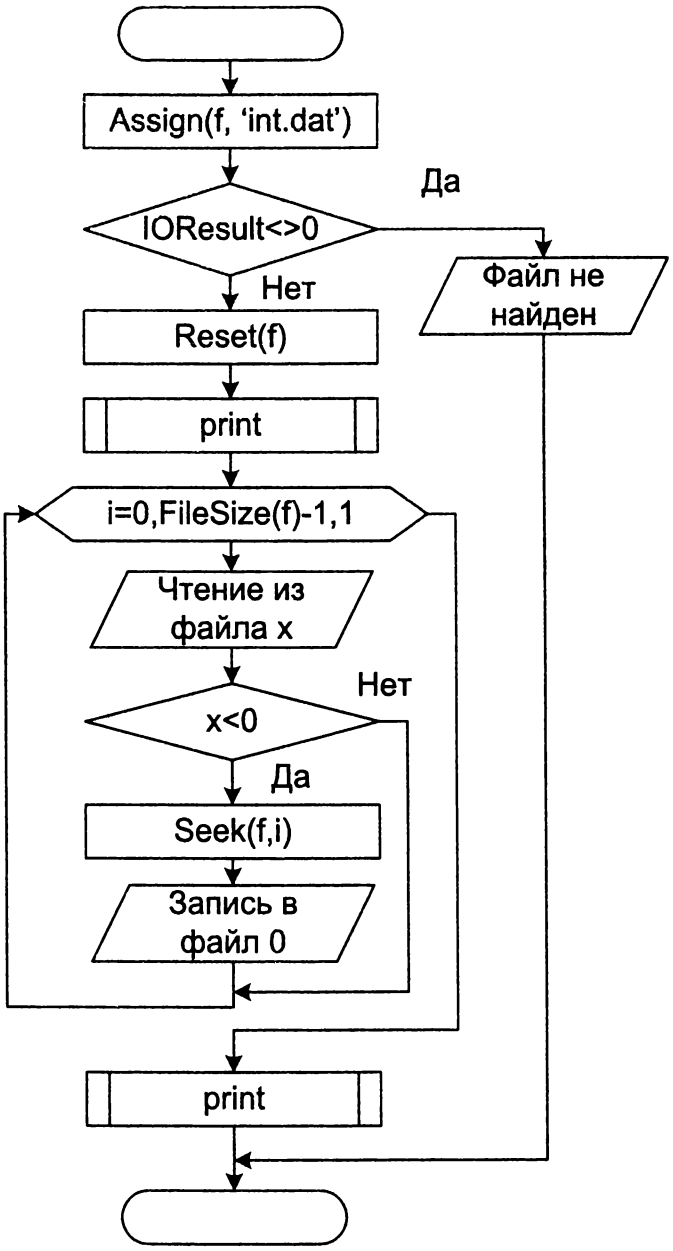

```
program vvod;
  var f: file of real;
      x: real;
      kon: char;
procedure print;
  var k,i: integer;
      x: real;
begin
   Assign(f,'vvod.dat');
   if IOResult<>0 then
   {если файла нет, то создать}
       Rewrite(f);
     else
     begin
   {если файл есть, открыть его}
       Reset(f);
   {поставить указатель в конец файла}
       Seek(f, FileSize(f);
     end;
   {цикл для ввода и записи данных}
    repeat
     write('x=');
     readln(x);
     write(f,x);
     write('Для окончания ввода введите Y');
     writeln('иначе любой символ');
     readln(kon);
    {Работа цикла, пока не введено 'Y'}
   until kon='Y';
   {Чтение файла и вывод на экран}
   print;


    end.
```

Пример 3

Дан типизированный файл, содержащий целые числа. Заменить в файле все отрицательные числа нулями. Для решения этой задачи потребуется типизированный файл, который должен быть создан заранее в отдельной программе.

Исходные данные: типизированный файл 'int.dat', содержащий целые числа.

Результат: типизированный файл 'int.dat', содержащий целые числа, с обновленными данными.



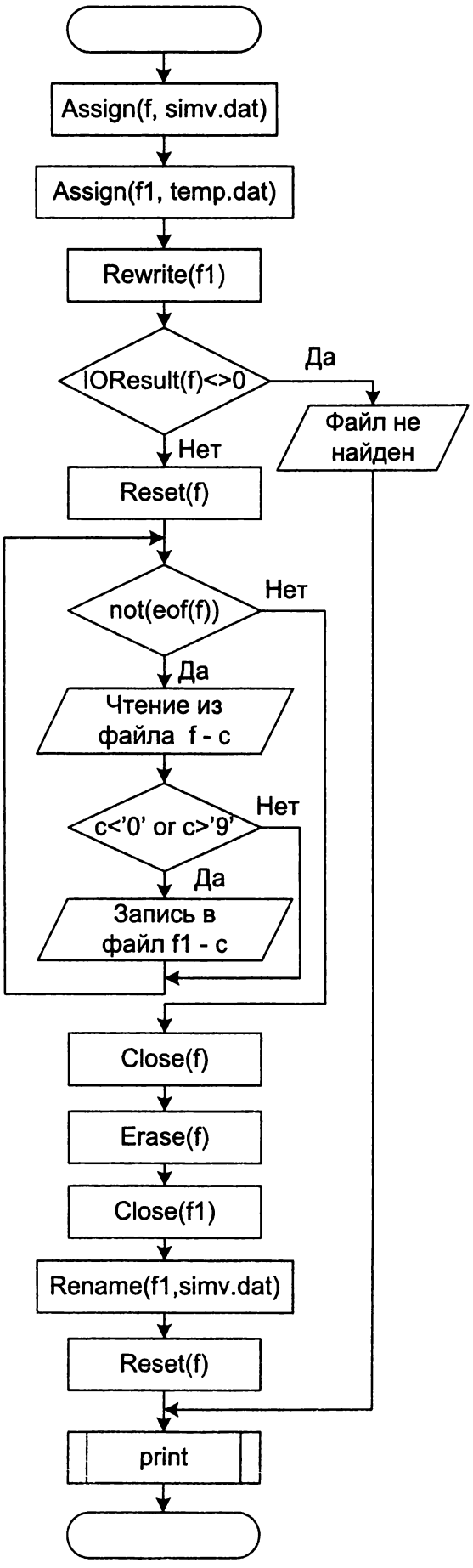
Пример 4

Дан типизированный файл, содержащий символы. Удалить из файла все символы цифр. Для решения этой задачи потребуется типизированный файл, который должен быть создан заранее в отдельной программе.

Исходные данные: типизированный файл ‘simv.dat’, содержащий целые числа.

Результат: типизированный файл ‘simv.dat’, содержащий символы, отличные от цифр.

В этом примере требуется удалить символы – значит, размер файла уменьшается. Это возможно только при использовании промежуточного файла, куда записываются допустимые символы. Затем данный файл переименовывается в первоначальный.




```
program udalen;
 var f,f1: file of char;
     c:char;
{процедура печати записей из файла}
procedure print;
    var i:integer; h:char;
begin
    For i:=0 to filesize(f)-1 do
      begin
         read(f,h);
         writeln(h);
      end;
end;
begin
{ связь с f файлом smv.dat}
  Assign(f, 'smv.dat');
{ связь с f1 файлом temp.dat}
  Assign(f1, 'temp.dat');
{создание файла f1}
  Rewrite(f1);
{проверка существования файла f}
  if IOResult(f)<>0 then
    {если файла нет, выйти из
программы}
                begin
                writeln('Файл не найден');
                 readln;
                 exit;
               end;
{открытие f для чтения}
    Reset(f);
    {пока не достигнут конец файла f}
    while not(eof(f)) do
       begin
          {считывание записи в c}
          read(f,c);
           {если c не цифра}
          if (c<'0') or (c>'9') then
             {запись c в f1}
              write(f1, c);
        end;
{конец цикла}
       {закрыть f}
       Close(f);
       {удалить f}
       Erase(f);
       {закрыть f1}
        Close(f1);
       {переименовать f1}
        Rename(f1,'simv.dat');
       {открыть для чтения}
         Reset(f);
       {распечатать записи из файла}
       print;

end.
```

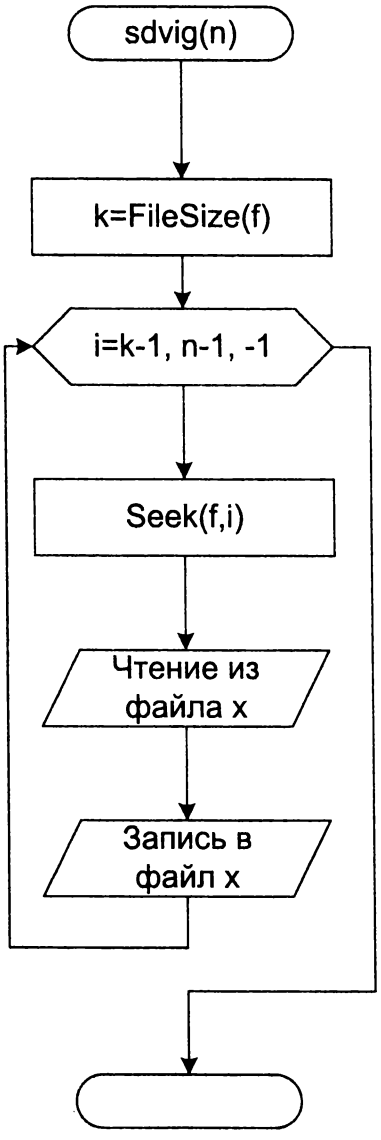
Пример 5

Дан типизированный файл, содержащий строки длиной 10 символов. Перед каждой строкой, начинающейся с заглавной английской буквы, вставить запись «название». Для решения этой задачи потребуется типизированный файл, который должен быть создан заранее в отдельной программе.

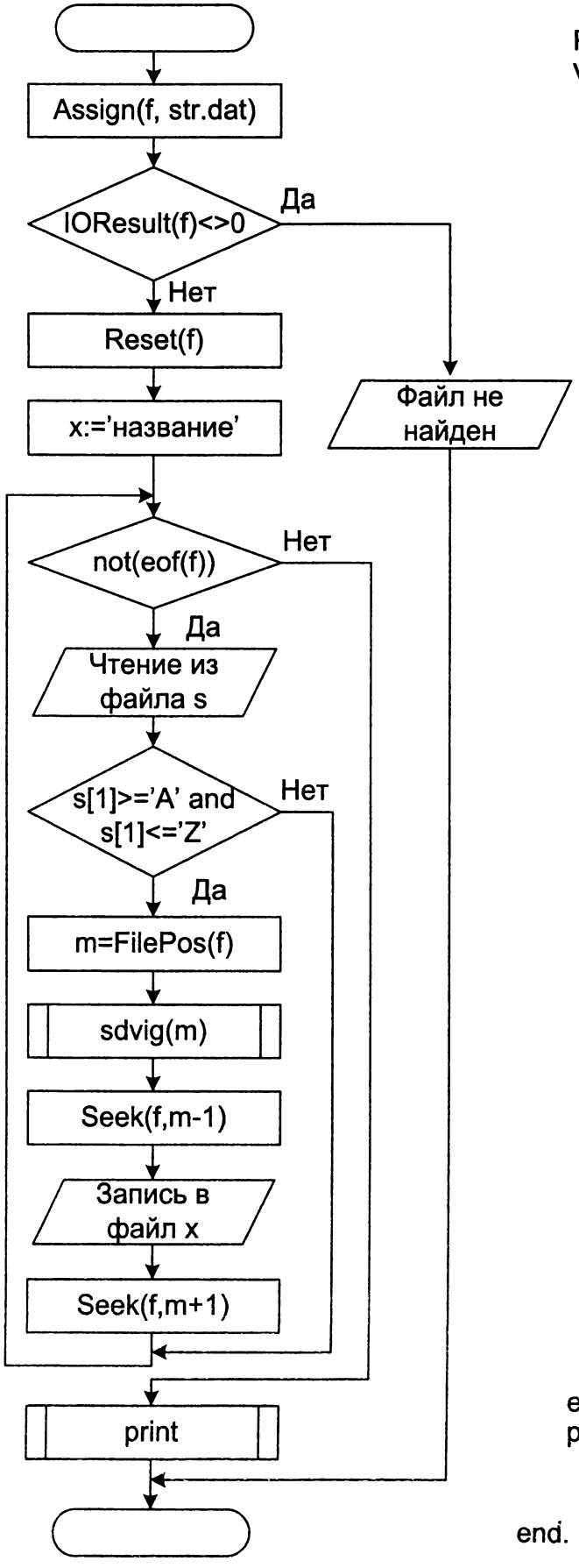
Исходные данные: типизированный файл ‘str.dat’, содержащий строки длиной не более 10 символов.

Результат: типизированный файл ‘str.dat’, содержащий строки длиной не более 20 символов, с обновленными данными.

Следует отметить, что в типизированном файле, содержащем строки, чтобы избежать ошибки, нужно указывать максимальное количество символов в строке.




```
procedure sdvig(n:integer);
  var i,k:integer;
      t:string[10];
begin
   k:=FileSize(f)

   For i:=k-1 dovnto n do
   begin
      Seek(f,i)

      Read(f,t)

      Write(f,t)

   end;

end;
```




```
Program tip_file;
Var f:file of string[10];
    s:string[10];
    x:string[10];
    m:integer;
    ...
 begin
    Assign(f,'str.dat');
    If OlResult(f)<>0 then
          begin
             writeln('Файл не найден');
          readln;
        exit;
      end
    else
       begin
         Rezet(f);
           x:='название';
           while not(eof(f)) do
              begin
                read(f,s);


       If (s[1]>='A') and (s[1]<='Z') then
            begin

             m:FilePos(f);

             sdvig(m);

              Seek(f,m-1);


             write(f,x);


              Seek(f,m+1);
          end;
    end;
end;
print;

end.
```

Контрольные вопросы

1. Какие процедуры используются для записи данных в типизированный файл?

2. Можно ли для просмотра данных в типизированном файле использовать цикл со счетчиком?

3. Почему в типизированном файле можно перемещать указатель на любую запись?

4. Можно ли создавать типизированный файл с помощью программы *Блокнот*?

5. Можно ли просматривать типизированный файл с помощью программы *Блокнот*?

Задания для лабораторной работы

1. Дан типизированный файл, содержащий целые числа. Переписать все отрицательные числа в начало файла.

2. Дан типизированный файл, содержащий строки. Заменить все строки, начинающиеся с гласной буквы, на последнюю строку в файле.

3. Дан типизированный файл, содержащий вещественные числа. Заменить все отрицательные числа на среднее значение.

4. Дан типизированный файл, содержащий целые числа. Удалить из файла все нули.

5. Дан типизированный файл, содержащий вещественные числа. Заменить каждую вторую запись на максимальное значение.

6. Дан типизированный файл, содкржащий символы. Все символы-цифры записать в конец файла.

7. Дан типизированный файл, состоящий из строк. Удалить строки, содержащие знаки препинания.

8. Дан типизированный файл, содержащий целые числа. Удвоить каждое положительное число.

9. Дан типизированный файл, содержащий вещественные числа. Добавить в конец файла минимальное и максимальное значения.

10. Дан типизированный файл, содержащий символы. Добавить в конец файла символ, имеющий самый маленький код из символов данного файла.

11. Дан типизированный файл, состоящий из символов. Заменить все строчные буквы заглавными.

12. Дан типизированный файл, содержащий целые числа. Переписать все положительные числа в конец файла.

Заключение

В данном учебном пособии рассмотрены основы алгоритмизации с иллюстрацией на языке программирования Pascal. Необходимо отметить, что акцент был сделан на разработке блок-схем алгоритмов, что позволяет использовать учебное пособие при изучении любых других языков программирования, так как блок-схемы являются универсальным средством записи алгоритмов: после разработки алгоритма с помощью блок-схемы остается только сопоставить блок с соответствующим оператором выбранного языка программирования.

Поскольку данное учебное пособие ориентировано на формирование у обучающихся знаний основ алгоритмизации и программирования, все задания, предложенные в нем, выполняются в консольном приложении соответствующего языка программирования. Следующим шагом является разработка приложения с помощью визуального программирования. На этом этапе появляются новые понятия: «формы», «визуальные компоненты», «событийное программирование» и т. д. Однако их полное понимание и эффективное использование новых приемов программирования невозможны без уверенного владения основами алгоритмизации, рассмотренными в данном учебном пособии.

Список литературы

1. *Абрамов В. Г.* Введение в язык Паскаль / В. Г. Абрамов, Н. П. Трифонов, Г. Н. Трифонова. Москва: Найка, 1988. 320 с.

2. *Вирт Н.* Алгоритмы и структура данных. Новая версия для Оберона / Н. Вирт. Санкт-Петербург: Невский Диалект, 2005. 352 с.

3. *Голицина О. Л.* Основы алгоритмизации и программирования / О. Л. Голицина, И. И. Попок. Москва: Форум: Инфра-М, 2002. 432 с.

4. *Кнут Д.* Искусство программирования: в 4 томах / Д. Кнут. Санкт-Петербург: Диалектика: Вильямс, 2007. Т. 1. Вып. 1: MMIX – RISC – компьютер для нового тысячелетия. 682 с.

5. *Рапаков Г. Г.* Turbo Pascal для студентов и школьников / Г. Г. Рапаков, С. Ю. Ржеуцкая. Санкт-Петербург: БХВ-Петербург, 2007. 352 с.

6. *Семакин И. Г.* Основы программирования / И. Г. Семакин, А. П. Шестаков. Москва: Академия, 2004. 432 с.

7. *Схемы* алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения [Электронный ресурс]: ГОСТ 19.701–90, ИСО 5807–85. Режим доступа: http://base.garant.ru/5369717/. (Единая система программной документации.)

Содержание

Учебное издание

Петров Сергей Борисович
Ширева Светлана Николаевна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебное пособие

Редактор О. Е. Мелкозерова
Компьютерная верстка Н. А. Ушениной

Печатается по постановлению
редакционно-издательского совета университета

Подписано в печать 29.12.11. Формат 70/108×16. Бумага для множ. аппаратов. Печать плоская. Усл. печ. л. 8,7. Уч.-изд. л. 9,0. Тираж 100 экз. Заказ № 52.
Издательство Российского государственного профессионально-педагогического университета. Екатеринбург, ул. Машиностроителей, 11.

Отпечатано ООО "ТРИКС"
Свердловская обл., г. Верхняя Пышма, ул. Феофанова, 4
www.printvp.ru