Министерство науки и высшего образования РФ
Московский государственный университет геодезии и картографии

А. М. Сёмов

Информатика: основы разработки алгоритмов и программ

Учебное пособие для студентов факультета дистанционных форм обучения МИИГАиК

Москва
2020

Рецензенты:

Князева М.Д., генеральный директор АНО Центр дополнительного образования «Будущим- космонавтам», к.т.н.

Чугреев И.Г., профессор, д.т.н. (МИИГАиК)

А. М. Сёмов

Информатика: Основы разработки алгоритмов и программ Учеб. пособие /

М. :МИИГАиК, 2020.-85 с

«Информатика: Основы разработки алгоритмов и программ» предназначено студентам факультета дистанционных форм обучения МИИГАиК. Оно может быть использовано для организации занятий по информатике и на других факультетах. Изложение теоретического материала сопровождается примерами, нацеленными на развитие у студентов и абитуриентов навыков решения конкретных задач, а также вопросами и упражнениями для самостоятельного выполнения.

1. Алгоритм и его свойства

1.1. Понятие алгоритма

Понятие алгоритма так же фундаментально для информатики, как и понятие информации. Поэтому в нем очень важно как следует разобраться.

Само слово “алгоритм” происходит от имени выдающегося математика средневекового Востока Мухаммеда Аль Хорезми (787-850гг.). Им были предложены приемы выполнения арифметических вычислений с многозначными числами (вам они хорошо знакомы из школьной математики). Позже в Европе эти приемы назвали алгоритмами, от Algorithm — латинского написания имени Аль Хорезми. В наше время алгоритм понимается шире, не ограничиваясь только арифметическими вычислениями.

Алгоритм — это последовательность команд, управляющих работой какого-либо объекта. Назовем его объектом управления. Им может быть как техническое устройство, так и живое существо. В дальнейшем будем называть его исполнителем алгоритма.

Алгоритмы арифметических вычислений сформулированы для исполнителя-человека. С таким же успехом можно назвать алгоритмами множество различных инструкций, предписывающих последовательность действий человека для выполнения какой-либо работы. Например, кулинарный рецепт — это алгоритм работы повара с целью приготовления блюда; инструкция по сборке машинки из деталей детского конструктора — алгоритм для ребенка; инструкция по использованию кухонного комбайна — алгоритм для домохозяйки.

Вы, наверное, никогда не задумывались над тем, какое количество алгоритмов вам известно. Жизненный опыт человека растет с увеличением числа освоенных им алгоритмов. Например, чтобы ребенок научился покупать в магазине хлеб, ему нужно сначала рассказать (а лучше показать), как это делается. Освоив “алгоритм покупки хлеба”, он в дальнейшем будет успешно выполнять эту работу.

Поиск выигрышной тактики, а, следовательно, и алгоритма несложной игры — интересная и полезная задача. Рассмотрим одну из таких игр, которая называется игрой Баше.

Играют двое. Перед ними 21 предмет, допустим, камни (также может быть 11, 16, 26 и т.д. т.е. 5k + 1 камней, k-натуральное число). Игроки берут камни по очереди. За один ход можно взять 1-2-3-4 камня. Проигрывает тот, кто забирает последний камень,

Имеется выигрышная тактика для игрока, берущего камни вторым. Она заключается в том, чтобы брать такое количество камней, которое дополняет число камней, взятых соперником на предыдущем ходе, до пяти. Этот алгоритм можно описать в виде последовательности команд

алг *Игра_Баше*

нач

1. Предоставить ход сопернику
2. Взять столько камней, чтобы в сумме с предыдущим ходом соперника получилось 5 камней
3. Если после этого для партнера остался один камень, то объявить о своем выигрыше, иначе после хода партнера вернуться к выполнению команды 2

кон

Игрок, строго следующий этому алгоритму, будет всегда выигрывать, даже если он не понимает, почему так происходит.

В приведенном примере используется символика учебного *Алгоритмического языка* (АЯ).

Из примера видно, что при записи алгоритма на АЯ в начале находится заголовок, начинающийся со *служебного слова* **алг** (сокращенное слово “алгоритм”). Затем указывается название алгоритма, которое автор придумывает сам. Следующая часть называется *телом алгоритма*. Она начинается служебным словом **нач** (начало) и заканчивается словом кон (конец). Тело алгоритма представляет собой последовательность команд для исполнителя.

Здесь и в дальнейшем служебные слова в алгоритмах на Алгоритмическом языке будут записываться жирным шрифтом. В языках программирования (как и в АЯ) служебными называются слова, для которых определен однозначный смысл.

Всякий алгоритм составляется в расчете на конкретного исполнителя с учетом его возможностей. Для того, чтобы алгоритм был выполним, нельзя включать в него команды, которые исполнитель не в состоянии выполнить. Нельзя повару поручать работу токаря, какая бы подробная инструкция ему не давалась. У каждого исполнителя имеется свой перечень команд, которые он может исполнить. Такой перечень называется *системой команд исполнителя алгоритмов (СКИ).*

1.2. Свойства алгоритма

Алгоритм, составленный для конкретного исполнителя, должен включать только те команды, которые входят в систему команд исполнителя. Это свойство алгоритма называется ***понятностью***.

Алгоритм должен представлять решение задачи в виде последовательного выполнения отдельных простых шагов (этапов). Это свойство алгоритма называется ***дискретностью***

Алгоритм не должен быть рассчитан на принятие каких-либо самостоятельных решений исполнителем. Каждая команда алгоритма должна однозначно определять действие исполнителя. Это свойство алгоритма называется ***точностью*** (или ***определенностью***).

Исполнение алгоритма должно завершиться за конечное число шагов.

Это свойство алгоритма называется ***конечностью*** (или ***результативностью***) алгоритма.

Алгоритм решения задачи должен разрабатываться в общем виде. Он должен быть применим для целого класса задач, которые отличаются лишь исходными данными, которые нужно подавать на вход алгоритма, чтобы он привел к решению конкретной задачи. Это свойство алгоритма называется ***массовостью***.

Для успешного выполнения любой работы мало иметь ее алгоритм. Всегда требуются еще какие-то *исходные данные, с* которыми будет работать исполнитель (продукты для приготовления блюда, детали для сбора технического устройства и т.п.). Исполнителю, решающему математическую задачу, требуется исходная числовая информация. Задача всегда формулируется так: *дана исходная информация, требуется получить какой-то результат*. В математике вы привыкли в таком виде записывать условия задач.

Например, Дано: катеты прямоугольного треугольника: a = 3см; b = 4см. Найти гипотенузу c. Алгоритм решения той задачи можно представить в таком виде:

алг Гипотенуза

нач

1. Возвести a в квадрат,
2. Возвести b в квадрат.
3. Сложить результаты действий 1 и 2,
4. Вычислить квадратный корень из результата 3-го действия и принять его за значение c.

кон.

Каждую из этих команд может выполнить любой человек, знающий основы математики, следовательно, они входят в его систему команд.

Приступая к решению любой задачи, нужно сначала собрать все необходимые для ее решения данные.

Еще пример: для поиска номера телефона нужного вам человека исходными данными являются: фамилия, инициалы человека и телефонная книга (точнее, информация, заключенная в телефонную книгу). Однако этого может оказаться недостаточно. Например, вы ищете телефон Смирнова А.И. и обнаруживаете, что в книге пять строк с фамилией Смирнов А.И.

Ваши исходные данные оказались *неполными* для точного решения задачи (вместо одного телефона вы получили пять). Оказалось, что нужно знать еще домашний адрес.

Набор: “фамилия — инициалы — телефонный справочник — адрес” является *полным набором данных* в этой ситуации.

Только имея полный набор данных, можно точно решить задачу. Если исходные данные неполные, то задачу либо совсем нельзя решить{ничего нельзя узнать про гипотенузу по одному катету), либо получается неоднозначное решение (пять номеров телефонов).

В задачах управления физическими объектами (автомобиль, самолет, станок и т.п.) исходными данными является информация о состоянии объекта управления, об обстановке, его окружающей.

Обобщая все сказанное, сформулируем определение алгоритма.

Алгоритм — понятное и точное предписание исполнителю выполнить конечную последовательность команд, приводящую от исходных данных к искомому результату.

Если алгоритм обладает перечисленными выше свойствами, то работа по нему будет производиться исполнителем формально, (то есть без всяких элементов творчества с его стороны). На этом основана работа программно-управляемых исполнителей-автоматов, например, промышленных роботов. Робот-манипулятор может выполнять работу токаря, если он умеет делать все операции токаря (включать станок, закреплять резец, перемещать резец, замерять изделие). От исполнителя не требуется понимание сущности алгоритма, он должен лишь точно выполнять команды, не нарушая их последовательности.

А что такое программа? Отличается ли чем-то программа от алгоритма? *Программа* это алгоритм, записанный на языке исполнителя. Иначе можно сказать так: алгоритм и программа не отличаются по содержанию, но могут отличаться по форме.

Для алгоритма строго не определяется форма его представления. Алгоритм можно изобразить графически, можно — словесно, можно — какими-нибудь специальными значками, понятными только его автору. Но программа должна быть записана на языке исполнителя.

2. Алгоритмы работы с величинами для ЭВМ

Алгоритм составляется для конкретного исполнителя. Теперь в качестве исполнителя мы будем рассматривать компьютер, оснащенный системой программирования на определенном языке. В качестве такого языка используем учебный алгоритмический язык (АЯ).

Компьютер-исполнитель работает с определенными *данными* по определенной *системе команд*.

2.1. Типы данных

Данные. Компьютер работает с информацией, хранящейся в его памяти. Отдельный информационный объект (число, символ, строка, таблица и пр.) называется *величиной.* Всякая обрабатываемая программой величина занимает свое место (поле) в памяти ЭВМ. Значение величины — это информация.

Существуют три основных типа величин, с которыми работает компьютер: *числовой, символьный* и *логический.* Числовые величины, делятся на переменные и константы (постоянные). Например, в формуле $(a^2-2ab+b^2)$, где *a, b* — переменные, а *2* - константа.

Константы записываются в алгоритмах своими десятичными значениями, например: 23, 3.5, 34. Значение константы хранится в выделенной под нее ячейке памяти и остается неизменным в течение работы программы.

Переменные в программировании, как и в математике, обозначаются символическими именами. Эти имена называют *идентификаторами* (от глагола “идентифицировать”, что значит обозначать, символизировать). Идентификатор может быть одной буквой, множеством букв, сочетанием букв и цифр. Как правило, употребляются буквы только латинского алфавита и первый символ в идентификаторе — буква. Примеры идентификаторов: *A, X,* S3, *prim, r25* и т.п.

2.2. Система команд (алгоритмических действий) ЭВМ.

Всякий алгоритм строится исходя из системы команд исполнителя, для которого он предназначен. Независимо от того, на каком языке программирования будет написана программа, алгоритм работы с величинами составляется из следующих команд:

- ***присваивание***
- ***ввод***
- ***вывод***
- ***ветвление***
- ***цикл***

· ***обращение к вспомогательному алгоритму***

Команда присваивания — одна из основных команд в алгоритмах работы с величинами. Записывать ее мы будем так: <переменная> **:=** < выражение>. Значок ":=" читается "присвоить".

Например: Z := X + Y.

Компьютер сначала вычисляет выражение в правой части, затем результат присваивает переменной, стоящей слева от знака " := ". Если до выполнения этой команды содержимое ячеек, соответствующих переменным X, Y, Z, было, например, таким: X = 1, Y = 2, Z = – (прочерк в ячейке *Z* обозначает, что начальное число в ней может быть любым), то после выполнения команды станет следующим: *X =1, Y = 2, Z = 3*.

Если слева от знака присваивания стоит числовая переменная, а справа — математическое выражение, то такую команду называют *арифметической командой присваивания,* а выражение — арифметическим. В частном случае арифметическое выражение может быть представлено одной переменной или одной константой. Например: *X: = 5; Y: = X*.

Значения переменных, являющихся исходными данными решаемой задачи, как правило, задаются вводом.

Команда ввода в описаниях алгоритмов будет выглядеть так:

ввод <список переменных>.

Например: **ввод** A, B, C

На современных компьютерах ввод чаще всего выполняется в режиме диалога с пользователем. По команде ввода компьютер прерывает выполнение программы и ждет действий пользователя. Пользователь должен набрать на клавиатуре вводимые значения переменных и нажать клавишу <ВВОД>. Введенные значения присвоятся соответствующим переменным из списка ввода, и выполнение программы продолжится.

Вот схема выполнения приведенной выше команды.

1. Память до выполнения команды: A = ---B= ---C= ---
2. Процессор ЭВМ получил команду **ввод** A, B, C, прервал свою работу и ждет действий пользователя.
3. Пользователь набирает на клавиатуре: 1 3 5 и нажимает клавишу <ВВОД> или <Enter>. Память после выполнения команды: A = 1, B = 3, C = 5
4. Процессор переходит к выполнению следующей команды программы.

При выполнении пункта 3 вводимые числа должны быть отделены друг от друга какими-нибудь разделителями. Обычно это пробелы.

Из сказанного выше можно сделать вывод: *Переменные величины получают конкретные значения в результате выполнения команды присваивания или команды ввода*. Если переменной величине не присвоено никакого значения (или не введено), то она является неопределенной. Иначе говоря, ничего нельзя сказать, какое значение имеет эта переменная.

Результаты решения задачи сообщаются компьютером пользователю путем выполнения команды вывода.

Команда вывода в алгоритмах будет записываться так: **вывод** <список вывода>.

Например: **вывод** X1, X2.

По этой команде значения переменных X1 и X2 будут вынесены на устройство вывода (чаще всего это экран).

Команда ветвления имеет такой формат:

если <условие> **то**

<серия 1>

иначе

<серия 2>

кв

Служебное слово **кв** обозначает *конец ветвления*. <Серия> — это одна или несколько следующих друг за другом команд.

Если <условие> справедливо, то выполняется <серия 1>, в противном случае — <серия *2*>. Такое ветвление называется *полным*.

Команды серий следует записывать с отступом в два пробела от вертикали, с которой записывают служебные слова **если**, **то**, **иначе**, **кв**. В противном случае, читая текст не совсем примитивной программы, понять ее логику практически невозможно. *И как показывает опыт многих поколений студентов, программы, написанные без отступов, отладке не поддаются.*

Рассмотрим пример алгоритма с полным ветвлением:

Из двух заданных *неравных* чисел **a** и **b** вывести на печать наибольшее.

Алгоритм решения этой задачи можно представить в таком виде:

алг Печать1

нач

1. **Ввести** a и b
2. **если** a > b **то**
3. **вывод** a

иначе

4. **вывод** b

кв

кон.

В некоторых случаях используется *неполная* форма команды ветвления. Она имеет следующий формат:

если <условие> **то**

<серия>

кв

Здесь <серия> выполняется, если <условие> справедливо.

Предыдущую задачу можно решить по алгоритму с неполными ветвлениями:

алг Печать1

нач

1. **Ввести** a и b
2. **если** a > b **то**
3. **вывести** a

кв

4. **если** a < b **то**
5. **вывести** b

кв

кон.

Циклическим называют алгоритм, в котором повторяется выполнение последовательности команд (составляющих тело цикла), пока справедливо условие повторения тела цикла.

Команда цикла имеет такой формат:

пока <условие>, **повторять**

нц

<тело цикла>

кц

Служебное слово **нц** обозначает *начало цикла*, **кц** — *конец цикла*.

В приведенной команде проверяется <условие>. Если оно выполняется, то выполняется <тело цикла>. Затем происходит *возврат* на проверку условия. Если оно выполняется, то тело цикла повторяется. Если проверка условия дает отрицательный результат, то выполнение цикла завершится, и будет исполняться следующая команда программы.

Команды тела цикла следует записывать с отступом в два пробела от вертикали, с которой записывают служебные слова **пока, нц, кц.**

В циклических алгоритмах важно думать о том, чтобы цикл был конечным. Ситуация, при которой выполнение цикла никогда не заканчивается, называется *зацикливанием.*

Пример циклического алгоритма:

Найти сумму *S* последовательных натуральных чисел от заданного числа *m* до числа *n.*

алг Сумма;

нач

1. **ввод** *m, n*
2. s:=0 {обнуление сумматора}
3. a:=m {присвоение первого значения переменной a, в которую будем помещать значение каждого очередного слагаемого}
4. **пока** a <= n **повторять**

   нц

   s:= s + a

   a:= a + 1

   кц
5. **вывод** s

кон.

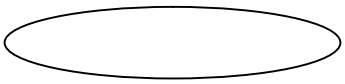
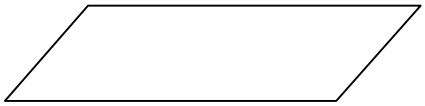
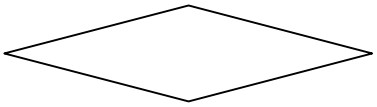
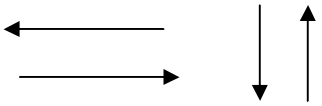
2.3. Блок – схемы алгоритмов

Начиная с 50-х годов, т.е. еще с эпохи ЭВМ первого поколения, программисты стали использовать графические схемы, изображающие алгоритмы, которые получили название блок-схем.

Блок-схема состоит из фигур (блоков), обозначающих отдельные действия исполнителя, и стрелок, соединяющих эти блоки и указывающих на последовательность их выполнения. Внутри каждого блока записывается выполняемое действие. Сама форма блока подсказывает характер операции, которую он обозначает. Для придания наглядности и единообразия схемам алгоритмов все графические элементы стандартизированы.

В табл. 1 приведены основные блочные символы.

Таблица 1

№	Блочный символ	Функция
1		Начало и конец алгоритма
2		Ввод и вывод данных
3		Простая команда - вычисление
4		Обращение к подпрограмме
5		Блок проверки условия
6		Стрелки передачи управления от блока к блоку. Определяют порядок выполнения алгоритма
7		Соединитель. В него приходят несколько стрелок управления, а выходит только одна

Алгоритм, как правило, составляют не из отдельных блоков, а из трех основных управляющих структур: СЛЕДОВАНИЕ, ВЕТВЛЕНИЕ, ЦИКЛ. В структуре следование действия выполняются последовательно одно за другим. Структура ветвления реализована командой ветвления.

Логика ее работы разобрана выше и наглядно изображается структурой, показанной на рис. 1.

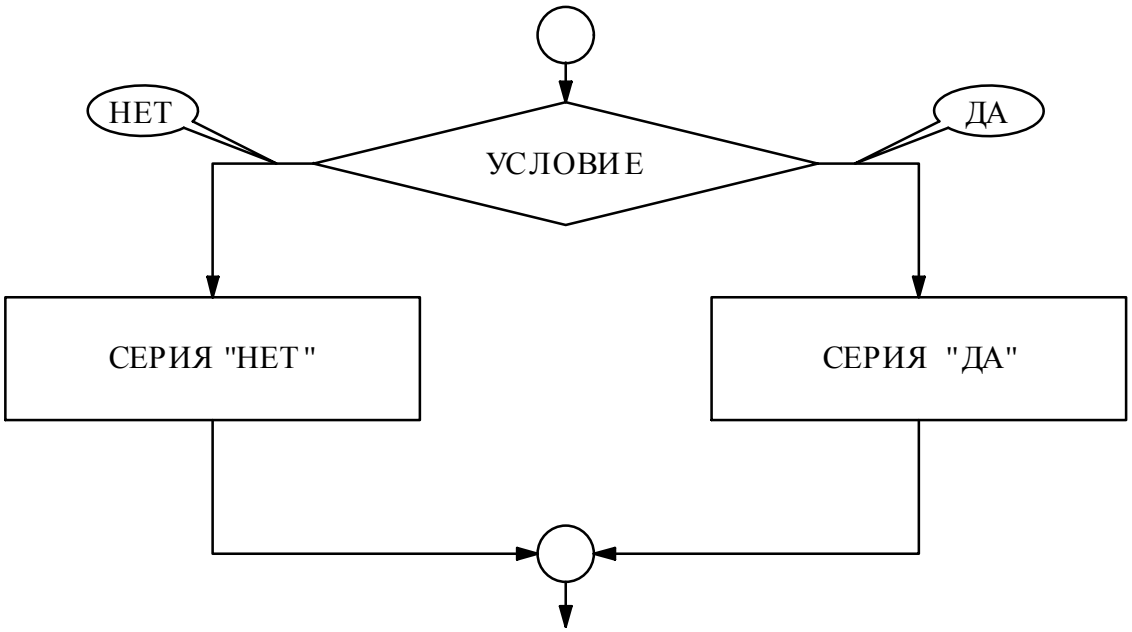


Рис. 1 Структура ветвление

Команда цикла, логика работы которой разобрана выше, изображается структурой, показанной на рис. 2. Такую структуру называют *циклом с предусловием* (так как условие предшествует телу цикла) или циклом Пока (While) .

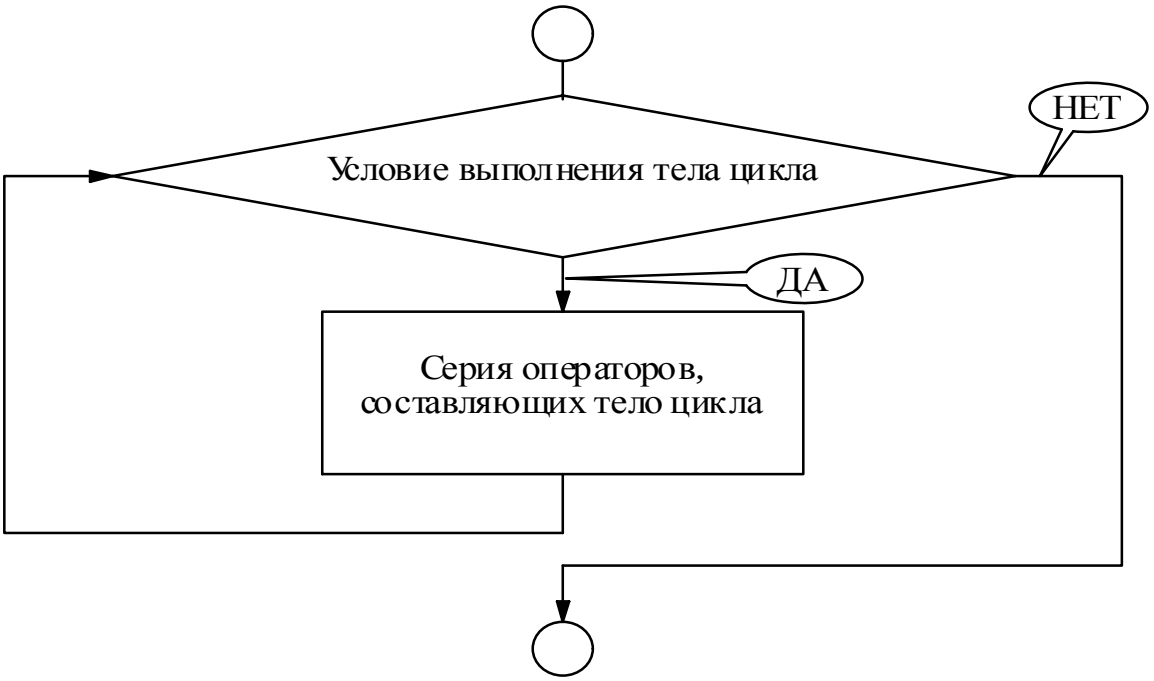


Рис. 2 Структура *цикл с предусловием*

Частным случаем цикла *пока* является цикл с параметром (его еще называют арифметический цикл или индексный цикл).

Блок-схема цикла с параметром представлена на Рис. 3:

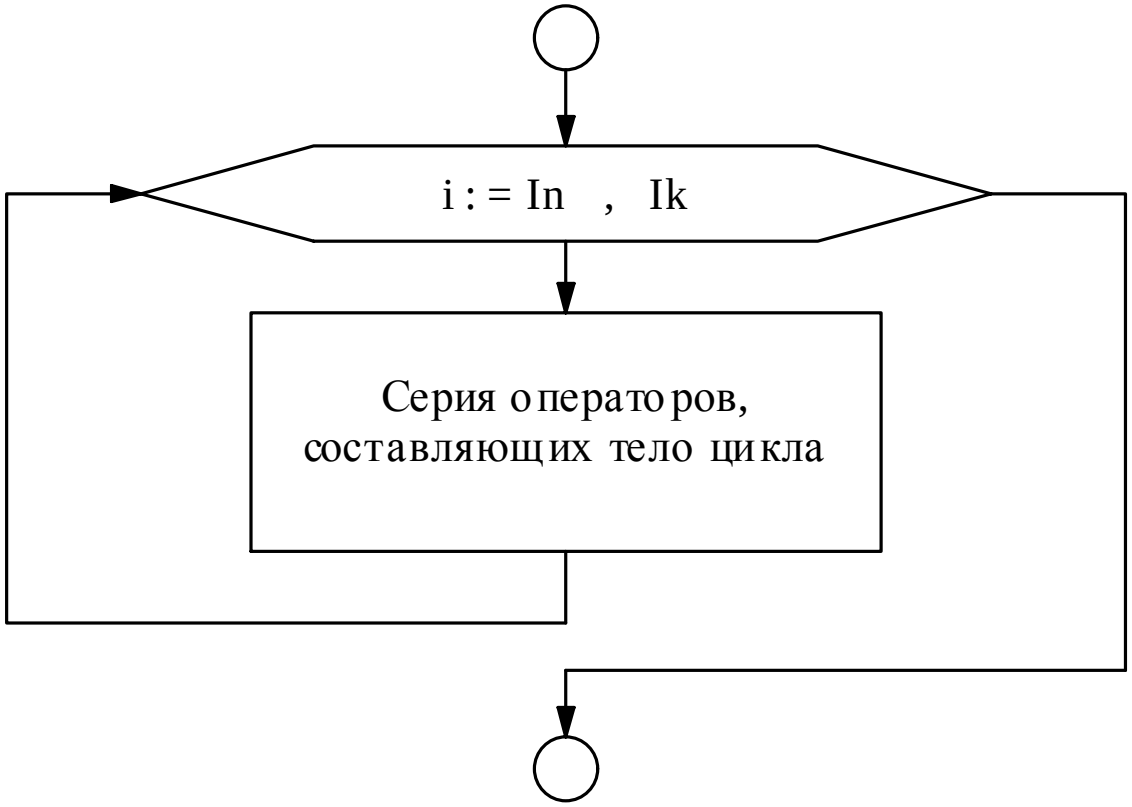


Рис. 3 Цикл с параметром

Цикл с параметром обеспечивает повторное выполнение тела цикла, пока целочисленный параметр *i* пробегает множество всех значений от начального (*In*) до конечного (*Ik*) шагом 1. Его запись на АЯ имеет вид:

для i **от** In **до** Ik, **повторять**

нц

<тело цикла>

кц

Команды тела цикла следует записывать с отступом в два пробела от вертикали, с которой записывают служебные слова **для, нц, кц.**

Пример циклического алгоритма. Найти сумму S натуральных чисел от заданного числа m до числа n. С использованием цикла с параметром алгоритм можно представить в виде:

алг Сумма1;

нач

1. **ввод** m , n

2. s:=0 {обнуление сумматора}

3. **для** i **от** m **до** n, **повторять**

нц

s:= s + i

кц

4. **вывод** s

кон.

2.4. Вопросы и упражнения

1. Что такое алгоритм? Откуда произошло это слово?
2. Что такое исполнитель алгоритма?
3. В чем состоят основные свойства алгоритма?
4. Назовите исполнителей следующих видов работы: уборка мусора во дворе, перевозка пассажиров, выдача заработной платы, прием экзаменов, сдача экзаменов, обучение детей в школе. Попробуйте сформулировать СКИ для каждого из этих исполнителей.
5. Определите полный набор данных для решения следующих задач обработки информации:

· вычисление стоимости покупок в магазине;

· вычисление суммы сдачи от данных вами продавцу денег;

· определение времени показа по телевизору интересующего вас фильма;

· вычисление площади треугольника;

· определение времени падения кирпича с крыши дома;

· определение месячной платы за расход электроэнергии;

· перевод русского текста на итальянский язык;

· перевод итальянского текста на русский язык.

6. Попробуйте сформулировать алгоритмы обработки информации для заданий из пункта 5, если исполнителем являетесь вы сами. Какие команды при этом вы должны уметь выполнять?
7. Какой формат имеет команда ветвления?
8. Какие действия исполнителя она определяет?
9. Чем отличается полное ветвление от неполного?
10. Что такое цикл? Как записывается команда цикла?
11. Что такое условие цикла?

12. Что такое тело цикла?
13. В каком случае происходит зацикливание алгоритма?
14. В чем состоят особенности цикла с параметром?
15. Что такое блок-схема?
16. Из каких блоков составляются блок-схемы (как они изображаются и что обозначают)?
17. Что обозначают стрелки на блок-схемах?
18. Нарисуйте блок-схему команды ветвления
    а) полного
    б) неполного
19. Нарисуйте блок-схему команды цикла ПОКА.
20. Нарисуйте блок-схему команды индексного цикла.

3. Программирование на алгоритмическом языке

Назначение программирования — разработка программ управления компьютером с целью решения различных информационных задач.

Для составления программ существуют разнообразные языки программирования. *Язык программирования* — это фиксированная система обозначений для описания алгоритмов и структур данных.

Для создания и исполнения на компьютере программы, написанной на языке программирования, используются *системы программирования*.

Система программирования — это программное обеспечение компьютера, предназначенное для разработки, отладки и исполнения программ, записанных на определенном языке программирования.

Разработка любой программы начинается с построения алгоритма решения задачи.

3.1. Линейные алгоритмы

Программы с линейной структурой составляются из операторов присваивания, ввода, вывода и обращения к подпрограммам.

Оператор присваивания *можно назвать основным в любом языке программирования*. Поговорим о нем более подробно.

Рассмотрим последовательность выполнения четырех команд присваивания, в которых участвуют две переменные величины *a* и *b*. В приведенной ниже табл. 2 напротив каждой команды указываются значения переменных, которые устанавливаются после ее выполнения. Такая таблица называется *трассировочной таблицей*, а процесс ее заполнения называется трассировкой алгоритма. Компьютер выполняет команды в порядке их записи в алгоритме.

Таблица 2

Команда	a	b
a:= 1	1	—
b:= 2*a	1	2
a:=b	2	2
b:= a + b	2	4

Прочерк в таблице обозначает неопределенное значение переменной. Конечные значения, которые получают переменные a и b, соответственно равны 2 и 4.

Этот пример иллюстрирует три основных свойства присваивания:

1. Пока переменной не присвоено значения, она остается неопределенной;
2. Значение, присвоенное переменной, сохраняется в ней вплоть до выполнения следующего присваивания нового значения этой переменной;
3. Новое значение, присвоенное переменной, заменяет ее предыдущее значение.

Рассмотрим еще один очень полезный алгоритм, с которым при программировании часто приходится встречаться. Даны две переменные величины X и Y. Требуется произвести между ними обмен значениями. Например, если первоначально было X = 1, Y = 2, то после обмена должно стать: X = 2, Y = 1.

Хорошим аналогом для решения такой задачи является следующая: даны два стакана, в первом — молоко, во втором — вода; требуется произвести обмен их содержимым. Всякому ясно, что в этом случае нужен дополнительный третий пустой стакан. Последовательность действий будет следующей:

1. Перелить из 1-го в 3-й;
2. Перелить из 2-го в 1-й;
3. Перелить из 3-го во 2-й.

Цель достигнута.

По аналогии, для обмена значениями двух переменных нужна третья дополнительная переменная. Назовем ее *Z*. Тогда задача решается последовательным выполнением трех операторов присваивания (Табл. 3). Начальные значения 1 и 2 для переменных X и Y задаются вводом.

Таблица 3

Команда	X	Y	Z
ввод X, Y	1	2	-
Z := X	1	2	1
X := Y	2	2	1
Y := Z	2	1	1
вывод X,Y	**2**	**1**	1

Действительно, в итоге переменные X и Y поменялись значениями. На экран будут выведены значения X и Y в таком порядке: 2, 1. В трассировочной таблице 3 выводимые значения выделены жирным шрифтом.

Аналогия со стаканами не совсем точна в том смысле, что при переливании из одного стакана в другой первый становится пустым, В результате же присваивания *X:=Y* переменная *Y*, стоящая справа, сохраняет свое значение.

И, наконец, рассмотрим пример составления алгоритма для решения следующей математической задачи: даны две обыкновенные дроби; получить дробь, являющуюся результатом их деления.

В школьном учебнике математики правила деления обыкновенных дробей описаны так:

1. Числитель первой дроби умножить на знаменатель второй.
2. Знаменатель первой дроби умножить на числитель второй
3. Записать дробь, числителем которой является результат выполнения пункта 1, а знаменателем — результат выполнения пункта 2,

В виде формулы это можно записать:

(a / b) **:** (c / d) = (a x d) / (b x c) = m / n

Теперь построим алгоритм деления дробей для компьютера. В этом алгоритме сохраним те же обозначения для переменных, которые использованы в записанной выше формуле. Исходными данными являются целочисленные переменные a, b, c, d. Результатом — также целые величины т и п.

Ниже алгоритм представлен на Алгоритмическом языке (АЯ). Полученный алгоритм имеет линейную структуру.

алг деление_дробей;

цел a, b, c, d;

нач

1. **ввод** a, b, c, d
2. m := a * d;
3. n := b * c ;
4. **вывод** m, n

кон.

В алгоритме на АЯ строка, стоящая после заголовка алгоритма, называется *описанием переменных.* Служебное слово **цел** означает целый тип. Величины этого типа могут принимать только целые значения.

3.2. Алгоритмы с ветвящейся структурой

Рассмотрим несколько задач, решение которых на компьютере получается с помощью ветвящихся алгоритмов,

Первая задача: *даны значения двух величин; выбрать большее из них.*

Пусть исходными данными являются переменные A и B. Их значения будут задаваться вводом. Значение большей из них должно быть присвоено переменной C и выведено на экран компьютера. Например, если A=5, B=8, то должно получиться: C=8.

А теперь запишем рассмотренный алгоритм на Алгоритмическом языке (АЯ). Во-первых, нужно решить вопрос о том, как описать переменные в этом алгоритме. Вспомним, что для всех переменных в алгоритме на АЯ необходимо указать их тип.

Переменные A, B, C — числовые величины. В этой задаче они могут принимать любые значения. В программировании числовые величины, которые могут иметь любые значения — целые, дробные — называются вещественными. Им ставится в соответствие *вещественный тип*. На алгоритмическом языке этот тип указывается служебным словом **вещ.**

```
алг БИД1
вещ a, b, c
нач
1.    Ввод a, b
2.    если a > b то
          c := a
      иначе
          c := b
      кв
3.    вывод c
кон.
```

Нетрудно понять смысл этого алгоритма. Если значение переменной A больше, чем B, то переменной C присвоится значение A, В противном случае, когда A < = B, переменной C присвоится значение B. (В качестве упражнения нарисуйте блок-схему этого алгоритма)

Условием, по которому разветвляется алгоритм, является отношение неравенства A>B. Такое отношение является *логическим выражением*. Если оно справедливо, то результатом будет логическая величина “истина” и выполнение алгоритма продолжится по ветви “да” или **то**; в противном случае логическое выражение примет значение “ложь” и выполнение алгоритма пойдет по ветви “нет” или **иначе.**

До выполнения на компьютере правильность алгоритма можно проверить путем заполнения трассировочной таблицы (Табл. 4).

Вот как будет выглядеть трассировка нашего алгоритма для исходных значений A=5, B=8.

Таблица 4

Шаг	Операция	A	B	C	Проверка условия
1	**Ввод** A,B	5	8		
2	A>B	5	8		5>8, нет (ложь)
3	C:=B	5	8	8	
4	**вывод** C	5	8	8	

Ветвление является *структурной командой.* Её исполнение происходит в несколько шагов: проверка условия (выполнение логического выражения) и выполнение команд на одной из ветвей “да” или “нет”. Поэтому в трассировочной таблице записываются не команды алгоритма, а отдельные операции, выполняемые компьютером на каждом шаге,

В рассматриваемом алгоритме используется *полное ветвление.* Эту же самую задачу можно решить, применяя структурную команду *неполного ветвления.*

Запишем такой алгоритм на Алгоритмическом языке.

алг БИД2
вещ a,b,c
нач
1. **Ввод** a, b
2. c:=a
3. **если** b > a **то**
4. c := b
кв
5. **вывод** c

кон.

Блок-схема такого алгоритма показана на рис. 4.

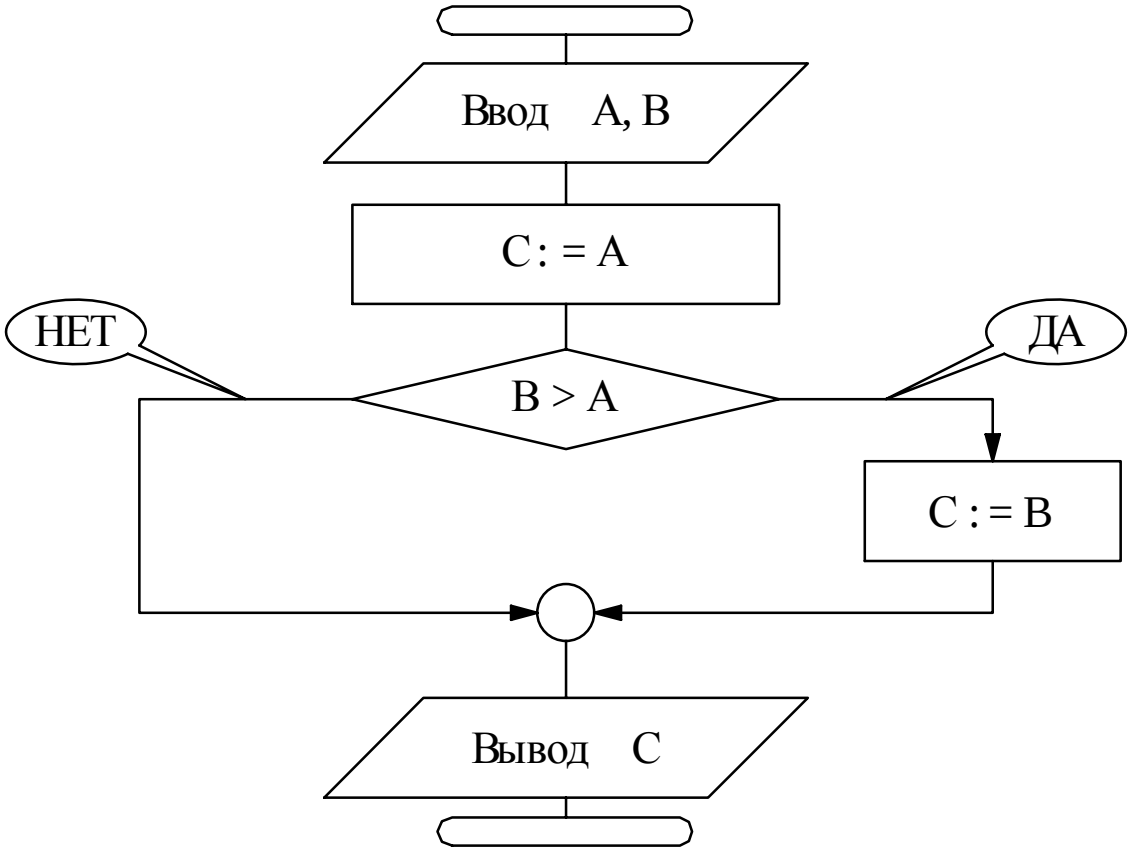


Рис. 4 Блок-схема алгоритма БИД2

Выполните самостоятельно трассировку этого алгоритма для вариантов

1. A=0.2, B=0.3;
2. A=7, B=4;
3. A=5, B=5.

Для программирования характерно то, что одна и та же задача может быть решена с помощью разных алгоритмов. И чем сложнее задача, тем больше можно придумать различных алгоритмов ее решения. Для больших задач (производственных, научных) практически невозможно точное совпадение алгоритмов, составленных разными программистами.

Следующая задача: *упорядочить и вывести значения двух переменных* X и Y *по возрастанию.* Смысл этой задачи следующий: если для исходных значений переменных

справедливо отношение X<Y (например, X=l, Y=2), то оставить их без изменения; если же X>Y (например, X=2, Y=l), то выполнить обмен значениями. Вывести X и Y.

Алгоритм обмена значениями двух переменных был рассмотрен в предыдущем параграфе. Вспомним, что для обмена нужна третья вспомогательная переменная.

В алгоритме решения данной задачи используется неполное ветвление. Алгоритм на АЯ имеют следующий вид

алг сортировка

вещ x,y,c

нач

1. **Ввод** x,y
2. **если** x > y **то**

   c := x

   x := y

   y := c

   кв
3. **вывод** x,y

кон.

Здесь роль вспомогательной переменной для обмена выполняет *С*.

3.3. Вопросы и упражнения

1. Что такое величина?
2. Чем отличаются переменные и постоянные величины?
3. Чем определяется значение величины?
4. Какие существуют основные типы величин в программировании?
5. Как записывается арифметическая команда присваивания?
6. Что такое ввод? Как записывается команда ввода?
7. Что такое вывод? Как записывается команда вывода?.
8. Что такое программирование?
9. Нарисуйте блок-схему алгоритма нахождения большего значения из двух величин.
10. Почему отношение неравенства можно назвать логическим выражением?
11. В каком случае для числовой переменной следует указывать тип целый, в каком — вещественный?

12. Опишите алгоритм (в виде блок-схемы и на АЯ) нахождения меньшего значения из двух.

13. Опишите алгоритм (в виде блок-схемы и на АЯ) нахождения меньшего значения из трех заданных.

14. Определите, какая задача решается по следующему алгоритму:

```
алг Задача-6
вещ X
нач
   вывод ‘Введите число не равное нулю’
   ввод  X
   если X<0 то
      вывод 'отрицательное число'
   иначе
      вывод 'положительное число или ноль'
   кв
кон
```

15. Составьте алгоритм, по которому на компьютере будет происходить следующее: в переменную S вводится возраст Саши, в переменную M вводится возраст Маши. В качестве результата на экран выводится фраза “Саша старше Маши” или “Маша старше Саши” (предполагаем, что кто-нибудь из них обязательно старше).

16. Решите предыдущую задачу, учитывая возможность одинакового возраста Саши и Маши. В таком случае будет еще и ответ: “Саша и Маша ровесники”.

17. Составьте алгоритм упорядочения значений трех переменных по возрастанию, т.е. при любых исходных значениях *A, B, C*. Поменять их значения местами так, чтобы стало $A<B<C$. Проверить алгоритм трассировкой при разных вариантах значений исходных данных.

18. Из каких команд составляется линейный вычислительный алгоритм?

19. Что такое трассировка? Как она производится?

20. В каком случае значение переменной считается неопределенным?

21. Что происходит с предыдущим значением переменной после присваивания ей нового значения?

22. Как вы думаете, можно ли использовать в арифметическом выражении оператора присваивания неопределенную переменную? К каким последствиям это может привести?

23. Напишите на АЯ алгоритм сложения двух простых дробей {без сокращения дроби).

24. Напишите на АЯ алгоритм вычисления *y* по формуле: $y= (1 -x^2+5x^4)^2$.где *x* — данное целое число. Выполните трассировку алгоритма при *x=2.*

Чтобы усложнить вашу задачу вводятся следующие ограничения:

в арифметических выражениях можно использовать только операции сложения, вычитания и умножения;

выражение может содержать только одну арифметическую операцию.

25. Запишите алгоритм циклического обмена значениями трех переменных *A, B, C*. Схема циклического обмена такова: например, если до обмена было: *A=1,B=2.C=3*, то после обмена должно стать: *A=3, B=1, C=2.* Выполнить трассировку.

4. Язык Паскаль

4.1. Основные элементы языка

После того, как построен алгоритм решения задачи, составляется программа на определенном языке программирования.

Среди современных языков программирования одним иа самых популярных является язык *Паскаль*. Этот язык разработан в 1971 году и назван в честь Блеза Паскаля — французского ученого, изобретателя механической вычислительной машины. Автор языка Паскаль — швейцарский профессор Никлаус Вирт. Вы будете использовать PascalABC.net-современную версию Паскаля, которая включает в себя как подмножество классический Паскаль и Турбо-Паскаль (ТП), поэтому вы можете просто писать программы на Паскале (ТП) и реализовывать их в среде программирования PascalABC.net.

Паскаль — это универсальный язык программирования позволяющий решать самые разнообразные задачи обработки информации.

Команду алгоритма, записанную на языке программирования, принято называть *оператором*.

Программа на Паскале близка по своему виду к описанию алгоритма на Алгоритмическом языке. Сравните алгоритм решения уже знакомой вам задачи — деление простых дробей, с соответствующей программой на Паскале (табл.5).

Таблица 5

алг деление дробей **цел** a, b, c, d, m ,n **нач** **ввод** a, b, c, d m := a x d n := b x c **вывод** m, n **кон**	**Program** Division; **var** a,b,c,d,m,n : integer; **begin** **readln**(a,b,c,d); {Ввод} m:= a*d; {Числитель} n:= b*c; {Знаменатель} **write**(m, n) {Вывод} **end**.

Заголовок программы начинается со слова **Program** (программа), за которым следует произвольное имя, придуманное программистом: **Program** < имя программы>;

Раздел описания переменных начинается со слова **Var** (variables — переменные), за которым идет список имен переменных через запятую. Тип указывается после двоеточия. В стандарте языка Паскаль существует два типа числовых данных: *вещественный* и *целый*. Слово

integer обозначает целый тип (является идентификатором целого типа). Вещественный тип обозначается словом **real**. Например, раздел описания переменных может быть таким:

Var a, b : **integer**;
c, d : **real**;

Идентификаторы ***переменных*** составляются из латинских букв и цифр; первым символом обязательно должна быть буква,

Раздел операторов — основная часть программы. Начало и конец раздела операторов программы отмечаются служебными словами **begin** (начало) и **end** (конец). В самом конце программы ставится точка.

begin
< операторы >
end.

< операторы > ***программы следует записывать с отступом в два пробела от вертикали***, с которой записывают служебные слова **begin end.**

Ввод исходных данных с клавиатуры происходит по оператору **read** (read — читать) или **readln** (read line — читать строку). Формат операторов:

read (<список переменных>); или **readln**(<список переменных>);

При выполнении команды ввода компьютер ожидает действий пользователя. Пользователь набирает на клавиатуре значения переменных в том порядке, в каком они указаны в списке, отделяя их друг от друга пробелами. Одновременно с набором данных на клавиатуре они появляются на экране. В конце нажимается клавиша <ВВОД> или <Enter>. Разница в выполнении операторов **readln** и **read** состоит в том, что после выполнения ввода по оператору **readln** экранный курсор перемещается в начало новой строки, а по оператору **read** этого не происходит.

Вывод **результатов** происходит по оператору **write** (write — писать) или **writeln** (write line — писать в строку). Формат оператора:

write (<список вывода>); или **writeln**(<список вывода>);

Результаты выводятся на экран компьютера в порядке их перечисления в списке. Элементами списка вывода могут быть константы, переменные, выражения.

Разница в выполнении операторов **writeln** и **write** состоит в том, что после выполнения вывода по оператору **writeln** экранный курсор перемещается в начало новой строки, а по оператору **write** этого не происходит.

Арифметический оператор присваивания на Паскале имеет следующий формат:

<числовая переменная> : = <выражение>;

Арифметическое выражение может содержать числовые константы и переменные, знаки арифметических операций, круглые скобки. Кроме того, в арифметических выражениях могут присутствовать функции.

Знаки основных арифметических операций записываются так:

+ сложение

- вычитание

* умножение

/ деление

Запись арифметических выражений на Паскале похожа на обычную математическую запись. В отличие от математики, где часто пропускается знак умножения (например, пишут 2a), в Паскале этот знак пишется обязательно: 2*a. Например, математическое выражение на Паскале записывается так: a*a + b*b- 12*c.

Это же выражение можно записать иначе: SQR(a) + SQR(b) - 12*c. Здесь использована функция возведения в квадрат — SQR. Аргументы функций всегда пишутся в круглых скобках,

Последовательность выполнения операций определяется по их *приоритетам* (старшинству). К старшим операциям относятся умножение «*****», деление «/». Операции сложения и вычитания — младшие. В первую очередь выполняются старшие операции. Несколько операций одинакового старшинства, записанные подряд, выполняются в порядке их записи слева направо.

Круглые скобки в арифметических выражениях влияют на порядок выполнения операций. Как и в математике, в первую очередь выполняются операции в скобках. Если имеются несколько пар вложенных скобок, то сначала выполняются операции в самых внутренних скобках.

Необходимо строгое соблюдение правописания (синтаксиса) программы. В частности, в Паскале однозначно определено назначение знаков пунктуации.

Точка с запятой «**;**» ставится в конце заголовка программы и в конце каждого оператора программы. Точка с запятой является разделителем операторов. Перед словом **end** точку с запятой можно не ставить.

Запятая является разделителем элементов во всевозможных списках: список переменных в разделе описания, список вводимых и выводимых величин.

Строгий синтаксис в языке программирования необходим потому, что *компьютер является формальным исполнителем программы*. Если, разделителем в списке переменных должна быть запятая, то любой другой знак будет восприниматься как ошибка. ***Так как точка***

с запятой является разделителем операторов в Паскале, то в качестве оператора компьютер воспринимает всю часть текста программы от одной точки с запятой до другой. *Если программист забыл поставить «**;**» между какими-то двумя операторами, то компьютер будет принимать их за один с неизбежной ошибкой*.

В программу на Паскале можно вставлять комментарии. *Комментарий* — это пояснение к программе, которое записывается в фигурных скобках. В комментариях можно использовать русские буквы. На исполнение программы комментарий никак не влияет.

Заметим, что в Паскале нет различия между строчными и прописными буквами. Разница только внешняя. Например, для Паскаля тождественны следующие варианты записи слова: begin, Begin, BEGIN, BeGiN. Использование строчных или прописных букв — дело вкуса программиста.

4.2. Подробнее о типах данных

В Турбо-паскаль(ТП)-программе все переменные должны быть описаны. Описать переменную - это значит указать её тип. Понятие типа - одно из фундаментальных понятий ТП. Тип определяет, во-первых, способ внутреннего представления объекта в памяти ЭВМ и, во-вторых, действия, которые разрешается над ним выполнять.

Рассмотрим следующие простые типы:

INTEGER - целочисленные данные; во внутреннем представлении занимают 2 байта; диапазон возможных значений - от -32768 до +32767; данные представляются точно.

REAL - вещественные данные; занимают 6 байт; диапазон возможных значений модуля от 2.9E-39 до 1.7E+38; точность представления данных - 11...12 значащих цифр.

CHAR - символ, занимает 1 байт.

STRING - строка символов; занимает MAX+1 байт, где MAX-максимальное количество символов в строке.

BOOLEAN - логический тип; занимает 1 байт и имеет два значения: FALSE (ложь) и TRUE (истина).

Тип константы определяется способом записи ее значения, например:

```
Const
c1 = 17;
c2 = 5.2;
c3 = ‘A’;
c4 =’ 3.14’;
c5 = false;
```

Первая константа относится к типу INTEGER, вторая - к типу REAL, третья к CHAR, четвертая к STRING и последняя к BOOLEAN. Признаком, позволяющим отнести константу к INTEGER или к REAL, является отсутствие или наличие десятичной точки в ее значении. Разумеется, константы *C2 и C4* относятся к разным типам: *C2* к REAL (в константе есть десятичная точка), а *C4* к STRING (константа обрамлена апострофами).

В качестве значений объявляемых констант в ТП допускаются выражения. Выражения должны в этом случае содержать только константы, в том числе и ранее объявленные, а также знаки математических операций, скобки и стандартные функции. Например:

```
Const
  x = (127 * 3) div z;
  y = x + 17;
  ArraySize = 800 * SizeOf(Real);
```

При описании переменных за именем переменной (идентификатором) ставится двоеточие и кодовое слово – имя типа. Несколько однотипных переменных можно объединять в список, разделяя их запятыми. В начале раздела описания переменных должно стоять кодовое слово VAR (от англ. Variables - переменные).

Например:

```
Var
   sigma     : real;
   a, b, c, d : char;
   text1     : string15];
   text2     : string;
   flag      : boolean;
```

Как уже говорилось, тип данных определяет длину внутреннего представления соответствующих переменных. В частности, длина внутреннего представления переменных типа STRING (строка символов) зависит от максимального количества символов, которые могут составлять строку. В приведенном выше примере переменная *TEXT1* описана с указанием ее максимальной длины (15 символов), а в описании переменной *TEXT2* максимальная длина не указана и компилятор установит для нее предельно допустимую в ТП длину-255 символов.

Рассмотрим еще одну несложную программу. Ее назначение - ввести с клавиатуры два целых числа, найти отношение первого ко второму и вывести полученный результат на экран.

```
Program MySecondProgram;
{ Программа вводит два целых числа и выводит частное от  деления 1-го на 2-е }
var
```

```
    n1, n2 : integer;    {n1 и n2 – вводимые целые}
    x :  real;        {x – результат}
Begin
   write(‘n1=’); Readln(n1);      {вводится n1}
   write(‘n1=’); Readln(n1);      {вводится n2}
    x := n1/n2;                   {найти результат}
   writeln(‘n1/n2=’,x);           {вывести x}
End.
```

В программе указаны поясняющие комментарии. Комментарий в ТП - это произвольная последовательность любых символов, обрамленная фигурными скобками. Комментарий разрешается вставлять в любое место программы, где по смыслу может стоять пробел.

Для вычисления отношения введенных чисел используется один из основных операторов ТП – оператор присваивания. В его левой части всегда указывается имя переменной, правая часть представляет собой выражение того же типа, что и переменная. Пара символов ":=" означает "присвоить значение".

4.3. Преобразование типов и действия над ними

Тип переменной позволяет не только установить длину ее внутреннего представления, но и контролировать действия, которые осуществляются над ней в программе. В ТП почти невозможны неявные (автоматические) преобразования типов. Исключение сделано только в отношении констант и переменных типа INTEGER, которые разрешается использовать в выражениях типа REAL.

Рассмотрим пример:

```
var
  x : integer;
  y : real;
   …..;
  y := x + 2;
```

Если, переменные X и Y описаны, как показано в примере, то оператор будет синтаксически правильным, хотя справа от знака присваивания стоит целочисленное выражение, а слева – вещественная переменная - компилятор сделает необходимые преобразования автоматически.

В то же время оператор *x* := 2.0 будет неверным, так как автоматическое преобразование типа REAL к типу INTEGER в ТП запрещено.

Разумеется, запрет на автоматическое преобразование типов еще не означает, что в ТП нет средств преобразования данных. Для преобразования данных в языке существуют встроенные функции, которые получают в качестве параметра значение одного типа, а возвращают результат в виде значения другого типа. В частности, для преобразования от REAL к INTEGER имеются даже две встроенные функции такого рода: **ROUND** округляет REAL до ближайшего целого, а **TRUNC** усекает REAL путем отбрасывания дробной части.

Для преобразования данных типа CHAR (символ) в целое число используется функция ORD, обратное преобразование от INTEGER к CHAR – функция CHR.

4.4. Арифметические операции

В ТП есть все четыре арифметические операции над переменными REAL и INTEGER (Табл. 6).

Таблица 6

+	сложение
–	вычитание;
*	умножение;
/	деление вещественное;
div	деление целочисленное.
mod	получение остатка от целочисленного деления

Целочисленное деление DIV выполняется над целыми числами или переменными и отбрасывает дробную часть результата. Для данных типа INTEGER в ТП есть еще одна операция деления - MOD, т.е. получение остатка от целочисленного деления, например:

5 **mod** 2=1,

В ТП отсутствует операция возведения в степень, но существует встроенная функция **SQR**, возвращающая квадрат от значения параметра, причем тип результата определяется типом параметра. Пример оператора, вычисляющего значение переменной *Y*, равное квадрату значения переменной *X:*

Y := **SQR** (X);

В ТП отсутствует операция извлечения корня, но существует встроенная функция **SQRT**, возвращающая корень квадратный от значения параметра. Пример оператора, вычисляющего значение переменной *Y*, равное корню квадратному из значения переменной *X*:

Y := **SQRT** (X);

В ТП над символами и строками символов определена единственная операция – сцепление двух строк. Операция обозначается символом "+".

Например, программа:

```
Var   s : string;
Begin
   s := ‘Турбо’+’–’+’Паскаль’;
   Writeln(s);
End.
```

– напечатает строку "Турбо-Паскаль".

4.5. Программирование диалога с компьютером

Если вы исполняли рассмотренные выше программы на компьютере, то почувствовали определенное неудобство при работе с машиной. Во-первых, непонятно, когда машина начинает ожидать ввода данных, какие данные и в каком порядке нужно вводить (это ведь можно и забыть). Во-вторых, результаты получаются в виде чисел на экране, без всяких пояснений их смысла. Ясно, что люди между собой так не общаются. Любую программу составлять нужно так, чтобы ее исполнение имитировало диалог между компьютером и пользователем в понятной для человека форме. Прежде, чем начать составление программы, нужно продумать *сценарий* такого диалога.

Например, составим сценарий работы программы, вычисляющей сумму двух целых чисел. На экране компьютера последовательно должны появляться следующие строчки

(для примера предположим, что будем вводить числа 237 и 658):

Введите первое слагаемое: A= **237**

Введите второе слагаемое: B= **658**

A + *B* = **895**

Пока

Здесь курсивом записаны символы, которые выводит компьютер по программе, а прямым жирным шрифтом — символы, вводимые пользователем.

Любой вывод на экран происходит по оператору вывода, записанному в программе.

Следовательно, с помощью оператора вывода на экран выносятся не только результаты решения задачи, но и все элементы диалога со стороны компьютера.

Вот программа, которая реализует наш сценарий:

```
Program Summa;
Var A, B : integer;
```

```
Begin
   write('Введите первое слагаемое: A= ');
    readln(A);
   write('Введите второе слагаемое: B= ');
    readln(B);
    writeln;
    writeln('A + B = ',   A+B);
    writeln('Пока!')
end.
```

В этой программе используется возможность включать в список вывода символьные строки, заключенные в апострофы, и арифметические выражения. Выражение A+B сначала вычисляется, а потом полученное число выводится на экран. Конечно, для вычисления суммы можно было написать отдельный оператор присваивания, но можно и так, как в этом примере,

Еще обратите внимание на оператор **writeln** без списка вывода. Он обеспечивает пропуск строки на экране.

При вводе вещественных чисел *целая и дробная часть числа отделяется десятичной точкой*

Составляя программу, вы сами организуете интерфейс компьютера с пользователем вашей программы. Этот *интерфейс обязательно должен быть дружественным*. Содержание диалога должно быть ясным и, непременно, вежливым!

4.6. Вопросы и упражнения

1. Когда появился язык Паскаль и кто его автор?
2. Как записывается заголовок программы на Паскале?
3. Как записывается раздел описания переменных?
4. С какими типами числовых величин работает Паскаль?
5. Как записываются операторы ввода и вывода в Паскале?
6. Что такое оператор присваивания?
7. Как записываются арифметические выражения?
8. По каким правилам определяется порядок выполнения операций в арифметическом выражении?
9. Какая задача решается по следующей программе?

```
Program Test;
Var A, B, C: integer;
begin
   readln(A,B);
   C := (A+B)*(B-A)
   writeln( C);
end.
```

10. Составьте программу, которая вычисляет длину окружности,, площадь круга и объем шара заданного радиуса.

11. Составьте программу, вычисляющую периметр и площадь прямоугольного треугольника по двум катетам.

12. Составьте программу, вычисляющую по координатам трех вершин некоторого треугольника, его площадь и периметр.

13. Составьте программу, вычисляющую дробную часть среднего геометрического двух заданных вещественных чисел. (Используйте функцию **TRUNC**).

14. Составьте программу, вычисляющую площадь кольца, если в качестве исходных данных задаются внешний диаметр кольца и отношение внешнего радиуса кольца к внутреннему.

4.7. Логические переменные и выражения

Для описания логической переменной используется стандартный тип ТП BOOLEAN. Переменные такого типа могут принимать только два значения – True(истина), или False(ложь). True и False являются логическими константами ТП. Логические переменные используются в логических выражениях.

Рассмотрим пример программы:

```
Var
 l : boolean;  {Описание логической переменной }
Begin
   l := True;   { Логический оператор присваивания }
  WriteLn('Значение логической переменной l=  ',l );
            {Вывод значения логической переменной}
End.
```

Логическим переменным значение может быть присвоено только в логическом операторе присваивания. В логических операторах присваивания справа от операции присваивания может располагаться любое логическое выражения, а слева должна стоять логическая переменная.

Оператор **Read (ReadLn)** для ввода значения логической переменной *не используется*. В тоже время, для вывода значения логической переменной *используется* оператор **Write (WriteLn)**.

Над данными, типа REAL, INTEGER, CHAR, STRING, BOOLEAN определены следующие ***операции отношения*** (сравнения) (Табл. 7).

Таблица 7.

>	**больше**
<	**меньше**
>=	**больше или равно**
<=	**меньше или равно**
=	**равно**
<>	**не равно**

В операциях сравнения должны участвовать однотипные операнды. Исключение сделано в отношении REAL и INTEGER, которые могут сравниваться друг с другом.

Результат применения операции отношения к любым операндам имеет тип BOOLEAN**.**

Сравнение двух строк осуществляется следующим образом. Символы строк сравниваются попарно друг с другом так, что первый символ первой строки сравнивается с первым символом второй строки, второй символ первой строки - со вторым символом второй и т.д. Символы сравниваются путем сравнения их кодов во внутреннем представлении. Если одна строка короче другой, недостающие коды символов заменяются нулми. Отношение первой несовпадающей друг с другом пары кодов символов и принимается за отношение двух строк.

Если в выражении присутствуют арифметические операции и операции отношения, то сначала выполняются арифметические операции, а затем операции отношения.

В логических выражениях, используются следующие ***логические операции***: операция OR (ИЛИ), операция XOR (исключающее ИЛИ)операция AND (И), операция NOT (НЕ), причем операции OR, XOR и AND связывают между собой два логических выражения, а операция NOT воздействует на одно логическое выражение.

Как работают логические операции, показывает таблица “ истинности ”(Табл. 8).

Таблица 8

Логическая операция	*Логическое выражение A*	*Логическое выражение B*	*Результат логической операции*
NOT	True		False
OR	True	True	True
	True	False	True
	False	True	True
	False	False	False
AND	True	True	True
	True	False	False
	False	True	False
	False	False	False
XOR	True	True	False
	True	False	True
	False	True	True
	False	False	False

При вычислении выражений любого типа приоритет вычислений определяется расставленными скобками, а при их отсутствии в порядке убывания приоритета логических операций (Табл. 9).

Таблица 9

Приоритет операций	*Операция*
1 высший	**not**
2 ниже	**and**
3 еще ниже	**or**
4 еще ниже	**xor**
5 еще ниже	=, < >, <, >, <=, > =

Рассмотрим примеры записи логических выражений:

1. (x>0) **and** (x <= 1);

2.(**not** (**sin**(x) < 1/3) **and** (x >0)) **or** ((**sqr** (y) < 3) **and** (y >0)).

Поскольку логические операции имеют более высокий приоритет, чем операции отношения, в сложных логических выражениях обычно необходимо расставлять скобки.

4.8. Оператор ветвления

В языке Паскаль имеется *оператор ветвления*. Другое его название — условный оператор. Формат полного оператора ветвления следующий:

if <логическое выражение> **then**

<оператор1>

else

<оператор2> **;**

Здесь **if** — если, **then** — то, **else** — иначе.

Сравните запись алгоритма БИД1 (см. стр.20) с соответствующей программой.BID1 на Паскале (Табл. 10).

Таблица 10

алг БИД1
вещ a, b,
нач
Ввод a, b
если a > b **то**
c := a
иначе
c := b
кв
вывод c
кон.
	program BID1;
	var a, b, c: real;
	begin
	readln (a, b);
	if a > b **then**
	c := a
	else
	c := b;
	writeln (c):
	end.

Очень похоже на перевод с русского языка на английский. Обратите внимание на следующее отличие: в программе нет специального служебного слова, обозначающего конец ветвления. Здесь *признаком конца оператора ветвления является* ***точка с запятой***. Поэтому *перед* **else** *точка с запятой не ставится*.

Простой формой логического выражения является *операция отношения*. Как и в Алгоритмическом языке, в Паскале допускаются все виды отношений (ниже указаны их знаки):

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

А теперь запрограммируем на Паскале алгоритм «БИД2» (см. стр.21), в котором использовано неполное ветвление (Табл. 11).

Таблица 11

алг БИД2
вещ a,b,c
нач
Ввод a, b
c:=a
если b > a **то**
c := b
кв
вывод c
кон.
	program BID2;
	var a, b, c:
	begin
	readln (a, b);
	c:= a;
	if b > a **then**
	c:=b**;**
	
	writeln (c);
	end.

Опять все очень похоже. Ветвь **else** («отрицательная» ветвь») в операторе ветвления может отсутствовать. Составим программу упорядочения значений двух переменных (Таблица 12).

Таблица 12

алг сортировка
вещ x, y, c
нач
Ввод x, y
если x > y **то**
c := x
x := y
y := c
кв
вывод x,y
кон.
	program BID2;
	var x, y, c;
	begin
	readln (x, y);
	if x > y **then**
	begin
	c := x;
	x := y; y := c;
	end;
	writeln (c):
	end.

Этот пример иллюстрирует следующее правило Паскаля: ***если на какой-то из ветвей оператора ветвления находится несколько последовательных операторов, то их нужно записывать между служебными словами*** **begin** *и* **end**.

Конструкция такого вида:

begin

<последовательность операторов>

end

называется *составным оператором*.

Следовательно, в описанной выше общей форме ветвления <оператор1> и <оператор2> могут быть простыми или составными операторами.

Обратите внимание на то, что ***перед словом*** **else** ***нельзя ставить точку с запятой***.

Алгоритмы ветвления могут иметь вложенную ветвящуюся структуру, которая программируется с помощью вложенных условных операторов. Например:

1) **If** <логическое выражение1 > **then**

<оператор 1>

else

if <логическое выражение 2> **then**

<оператор2>

else

<оператор 3>;

или

2) **If** <логическое выражение1 > **then**

If <логическое выражение2 > **then**

<оператор 1>

else

<оператор 2>

else

if <логическое выражение3> **then**

<оператор 3>

else

<оператор 4>;

или

3) **If** <логическое выражение1 > **then**

Begin

If <логическое выражение2 > **then**

<оператор 1>;

If <логическое выражение3 > **then**

<оператор 2>

else

<оператор 3>;

end

else

if <логическое выражение4> **then**

<оператор 4>

else

If <логическое выражение5 > **then**

<оператор 5>

else

<оператор 6>;

В качестве обязательного упражнения нарисуйте блок-схемы трех вложенных ветвлений, приведенных выше.

4.9. Оператор выбора

Оператор выбора является обощением условного оператора для случаев произвольного числа альтернатив. Оператор выбора позволяет выбрать для выполнения один из нескольких возможных операторов программы. Параметром, по которому осуществляется выбор, служит так называемый *ключ выбора* - выражение перечисляемого типа.(INTEGER, CHAR, BOOLEAN).

Структура оператора выбора такова:

CASE <ключ выбора > OF < список выбора > ELSE <оператор- альтернатива» END.

Здесь CASE, OF, ELSE, END - зарезервированные слова.

<ключ выбора > – выражение перечисляемого типа;

<список выбора > – одна или более конструкций вида – <константа выбора – значение ключа выбора > : < оператор >;

< константа выбора > – константа того же типа, что и выражение ключа выбора;

< оператор > - произвольный оператор ТП.

Оператор выбора работает следующим образом: сначала вычисляется значение выражения <ключ выбора>, а затем в последовательности операторов < список выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению ключа выбора. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора, управление передается оператору, стоящему за кодовым словом ELSE.

Рассмотрим простейший пример оператора выбора

```
Case Color of
    Red    :  x := y + 2;
   Yellow :  x := y – 2;
   Green   :  x := y
Else
    Writeln('Значение ключа выбора не совпадает с константами из списка выбора')
End;
```

Если значение переменной Color равно константе Red, то выполняется оператор *x := y* + 2 оператор выбора завершается. Аналогичные действия выполняются при значении переменной Color, равной Yellow или Green. Если значение Color не совпадает со списком (Red, Yellow, Green), то будет выполняться оператор, расположенные после слова ELSE.

Часть ELSE < оператор > можно опускать. Тогда при отсутствии в списке выбора нужной константы оператор выбора просто завершит свою работу.

Любому из операторов списка выбора может предшествовать не одна, а несколько констант выбора, разделенных запятыми. Например, следующая программа при вводе одного из символов *y* или *Y* выведет на экран слово "Да", а при вводе *n* или *N* - слово "Нет":

```
Var    ch  : char;
Begin
  readln(ch);
  case ch of
     'n','N'  : writeln('Нет');
     'y','Y' : writeln('Да');
```

```
    end;
End.
```

4.10. Вопросы и упражнения

1. Как программируется на Паскале полное и неполное ветвление?
2. Что такое составной оператор? В каких случаях составной оператор используется в операторе ветвления?
3. Что обозначает понятие «диалоговый характер программы»?
4. Какими средствами программируется диалог между пользователем и компьютером?
5. Что обозначает понятие «дружественный интерфейс»?
6. Составьте программы в соответствии с алгоритмами БИТ1 и БИТ2 из подраздела последовательные и вложенные ветвления.
7. Составьте программы в соответствии с алгоритмами БИД1 и БИД2 из раздела Алгоритмы с ветвящейся структурой
8. Постройте алгоритм и составьте программу, по которой будет реализован следующий сценарий: компьютер запрашивает номер дня недели, после ввода компьютер сообщает название этого дня. Например, если ввели 1, то выведется фраза «Это понедельник» и т.д.

4.11. Технология решения задач на ЭВМ. Программирование циклов

Вы научились составлять линейные и ветвящиеся программы на Паскале. Теперь нужно освоить программирование циклов. Снова будем учиться на примере конкретной задачи. Но, в отличие от предыдущих примеров, начнем с изложения общего подхода к программированию с целью решения конкретной задачи.

Часто задача, которую требуется решить, сформулирована не на математическом языке. Для решения на компьютере ее сначала нужно привести к форме математической или логической задачи, а потом уже программировать.

Работа по решению таких задач с использованием компьютера проходит через следующие этапы:

1. Постановка задачи.

2. Математическая формализация,
3. Построение алгоритма.
4. Составление программы на языке программирования.
5. Отладка и тестирование программы.
6. Проведение расчетов и анализ полученных результатов.

Эту последовательность называют *технологической цепочкой решения задачи на ЭВМ.*

В чистом виде программированием, то есть разработкой алгоритма и программы, здесь являются лишь 3-й, 4-й и 5-й этапы,

На этапе постановки задачи должно быть четко определено, что *дано* и что *требуется найти.*

Второй этап — математическая формализация. Здесь задача переводится на язык математических формул, уравнений, отношений. Далеко не всегда эти формулы очевидны. Нередко их приходится выводить самому или отыскивать в специальной литературе. Если решение задачи требует математического описания какого-то реального объекта, явления или процесса, то формализация равносильна получению соответствующей *математической модели.*

Третий этап — построение алгоритма. Вы знаете два способа описания алгоритмов: блок-схемы и Алгоритмический язык.

Первые три этапа – это работа без компьютера. Дальше следует программирование на определенном языке в определенной системе программирования и этап отладки и тестирования программы. Последний (шестой) этап- это использование уже разработанной программы в практических целях.

Проследим все этапы технологической цепочки на примере конкретной задачи.

· *Постановка задачи***.** Дано N кубиков, на которых написаны разные буквы. Сколько различных N-буквенных слов можно составить из этих кубиков (слова не обязательно должны иметь смысл)?

Искомую целочисленную величину обозначим буквой F, Тогда постановка задачи выглядит так

ДАНО: N кубиков, на каждом – только одна буква, все буквы разные.

НАЙТИ: F

· *Математическая формализация*. Получим расчетную формулу. При составлении слова в первую позицию можем поставить N вариантов буквы. Во вторую позицию для каждого варианта заполнения 1 – ой позиции можем поставить $(N-1)$ вариант буквы (любую букву, из оставшихся после заполнения первой позиции). Таким образом, первые две позиции

можно заполнить Nx(N - 1) способами. В третью позицию можем выбрать (*N* - 2) варианта буквы (любую букву, из оставшихся после заполнения первых двух позиций). Таким образом первые три позиции можно заполнить Nx(N - 1)x (N-2) способами и так далее. Для последней *N* – ой позиции останется ровно одна буква. Таким образом, формула для *F* имеет вид:

F = Nx(N - 1)x(N-2)x(N-3)x….x3x2x1.

Произведение *N* последовательных натуральных чисел, начиная с 1, носит название *факториал* числа *N* и обозначается ***N!***

Теперь вернемся к формулировке задачи. Если *N* обозначает количество букв, а *F* — количество слов из этих букв, то расчетная формула такая:

F = N!=lx2x..xN.

· *Построение алгоритма*. Поскольку алгоритм должен быть независимым от данного значения *N*, то его нельзя сделать линейным. Дело в том, что для разных *N* надо выполнить разное число умножений. В таком случае с изменением *N* линейная программа должна менять длину.

Алгоритм решения данной задачи будет *циклическим. С* циклическими алгоритмами вы уже познакомились,

Цикл — это команда исполнителю многократно повторить указанную последовательность команд. Рассмотрим алгоритм на АЯ.

```
алг   СЛОВА
цел   F,N,R
нач
   ввод N
   F:=l
   R:=1
   пока R <= N,   повторять
   нц
        F := F * R
        R := R + 1
    кц
    вывод F
кон
```

Здесь применена знакомая вам алгоритмическая структура *«цикл с предусловием»*. Выполняется она так: пока *истинно условие повторения цикла, повторяется выполнение тела цикла.*

Тело цикла составляют две команды присваивания, заключенные между служебными словами **нц** и **кц**. Условие цикла — это отношение $R<=N$ (R меньше или равно N).

В данном алгоритме переменная R выполняет роль множителя, значение которого меняется от 1 до N через единицу. Произведение накапливается в переменной F, начальное значение которой равно 1. Цикл заканчивается, когда R станет равно N+1. Это значение в произведение уже не попадет.

Для проверки правильности алгоритма построим трассировочную таблицу.

Следующая трассировочная таблица (Табл.13) составлена для случая N=3

Таблица 13

Шаг	Операция	N	F	R	Условие
1	ввод N	3	-	-	
2	F:=1		1	-	
3	R:=1			1	
4	R<N				1<=3,да
5	F:=FxR		1		
6	R:=R+1			2	
7	R<=N				2<=3. да
8	F:=FxR		2		
9	R:=R+1			3	
10	R<=N				3<=3.да
11	F:=FxR		6		
12	R:=R+1			4	
13	R<=N				4<=3 нет
14	вывод F		6		
15	конец				

Из этой таблицы хорошо видно, как менялись значения переменных. Новое значение, присвоенное переменной, стирает ее старое значение (здесь не повторяется запись значения переменной, если оно не изменяется; в таком виде таблица менее загромождена числами).

Последнее значение *F* равно 6. Оно выводится в качестве результата, Очевидно, что результат верный: 3!=6.

Составление программы. Чтобы составить программу решения нашей задачи, нужно научиться программировать циклы на Паскале.

4.12. Операторы циклов

Основной циклической структурой является *цикл с предусловием* (цикл-пока). С помощью этой структуры можно построить любой циклический алгоритм. Оператор цикла с предусловием в Паскале имеет следующий формат:

while <логическое выражение> **do** <оператор>.

Служебное слово **while** означает «пока», **do** — делать, выполнять. Оператор, стоящий после слова **do**, называется *телом цикла.* Тело цикла может быть простым или составным оператором, т.е. последовательностью операторов между служебными словами **begin** и **end**.

А теперь запрограммируем на Паскале алгоритм решения нашей задачи (добавим к нему организацию диалога).

```
Program Words;
var
  F, N, R: integer;
begin
   write('Введите число букв:  ');
    readln(N);
   F := l;
   R := l;
   while R<=N   do
      begin
         F := F * R;
         R := R + 1;
       end;
    writeln('Количество слов =  ', F);
end.
```

Снова бросается в глаза схожесть алгоритма на АЯ и программы на Паскале. Обратите внимание на то, что в Паскале нет специальных служебных слов для обозначения начала и конца цикла (так же как и конца ветвления). Во всех случаях, где это необходимо, используются слова **begin** и **end**.

А теперь, запишем тот же алгоритм с помощью индексного цикла Паскаля (цикла с параметром). Цикл с параметром обеспечивает повторное выполнение тела цикла, пока целочисленный параметр пробегает множество всех значений от начального (*In*) до конечного (*Ik*) шагом 1.Он имеет формат:

```
for i := In to Ik do
  begin
     <операторы
      тела
     цикла>
  end;
```

Обратите внимание на необходимость отступов от вертикали* For *и от вертикали* begin – end *при записи операторов тела цикла.

В Паскале используется еще и индексный цикл, в котором целочисленный параметр пробегает множество всех значений от начального (*In*) до конечного (*Ik*), при *In* большем *Ik*, шагом --1. Он имеет формат:

```
for i := In downto Ik do
   begin
      <операторы тела цикла>
   end;
```

Алгоритм показан в таблице 14.

Таблица 14

С оператором ***for – to - do***	*С оператором* ***for – downto - do***

Program Words1
Var
F, N, R: integer;
begin
write('Введите число букв: ');
readln(N);
F := l;
for i := 1 **to** N **do**
begin
F := F * i;
end;
writeln('Количество слов = ', F);
end.
	Program Words1;
	Var
	F, N, R: integer;
	begin
	write('Введите число букв: ');
	readln(N);
	F := l;
	for i := N **downto** 1 **do**
	begin
	F := F * i;
	end;
	writeln('Количество слов = ', F);
	end.

Отладка и тестирование*. Под отладкой программы понимается процесс испытания работы программы и исправления обнаруженных при этом ошибок.* Обнаружить ошибки, связанные с нарушением правил записи программы на Паскале (синтаксические и

семантические ошибки) помогает используемая *система программирования*. Пользователь получает сообщение об ошибке, исправляет ее и снова повторяет попытку исполнить программу.

Проверка на компьютере правильности алгоритма производится с помощью тестов. *Тест — это конкретный вариант значений исходных данных, для которого известен ожидаемый результат*. Прохождение теста — необходимое условие правильности программы. На тестах проверяется правильность реализации программой запланированного сценария.

Нашу программу, например, можно протестировать на значении N=6. На экране должно получиться:

Введите число букв: 6

Количество слов = 720

Проведение расчетов и анализ полученных результатов — этот этап технологической цепочки реализуется при разработке практически полезных (не учебных) программ. Например, «Программа расчета прогноза погоды». Ясно, что ею будут пользоваться длительное время, и правильность ее работы очень важна для практики. А поэтому в процессе эксплуатации эта программа может дорабатываться и совершенствоваться.

4.13. Алгоритм Евклида

Рассмотрим следующую задачу: требуется составить программу определения наибольшего общего делителя (НОД) двух натуральных чисел.

Вспомним математику. Наибольший общий делитель двух натуральных чисел — это самое большое натурально число, на которое они делятся нацело. Например, у чисел 12 и 18 имеются общие делители: 2, 3, 6. Наибольшим общим делителем является число 6. Это записывается так:

НОД(12,18)=6.

Обозначим исходные данные как M и N. Постановка задачи выглядит следующим образом:

ДАНО: M, N НАЙТИ НОД(M, N).

В данном случае какой-то дополнительной математической формализации не требуется. Сама постановка задачи носит формальный математический характер. Не существует формулы для вычисления НОД(M,N) по значениям M *и* N. Но зато достаточно давно, задолго до появления ЭВМ, был известен алгоритмический способ решения этой задачи. Называется он *алгоритмом Евклида*,

Идея этого алгоритма основана на том свойстве, что если *M>N*, то

НОД(*M,N*) = НОД(*M - N,N*).

Иначе говоря, НОД двух натуральных чисел равен НОД их положительной разности и меньшего числа.

Легко доказать это свойство. Пусть *K* — общий делитель *M и N* (*M>N*). Это значит, что *M=mK*, *N=nK*, где *m, n* — натуральные числа, причем *m>n*. Тогда *M-N=K(m-n)*, откуда следует, что *K* — делитель числа *M—N*. Значит, все общие делители чисел *M и N* являются делителями их разности *M-N*, в том числе и наибольший общий делитель. Отсюда:

НОД(*M,N*)= НОД(*M - N,N*).

Второе очевидное свойство: НОД(*M,M*) = *M*.

Для «ручного» счета алгоритм Евклида выглядит так:

1. если числа равны, то взять любое из них в качестве ответа, в противном случае продолжить выполнение алгоритма;
2. заменить большее число разностью большего и меньшего из чисел;
3. вернуться к выполнению пункта 1.

Запишем алгоритм на АЯ и на Паскале (Табл. 15).

Таблица 15

алг Евклид
цел M, N
нач
вывод 'Введите M и N'
ввод M, N
пока M не = N**, повторять**
нц
если M>N **то**
M:=M-N
иначе
N:=N-M
кв
кц
вывод 'НОД=', M
кон
	Program Evklid;
	var M, N: integer;
	begin
	writeln ('Введите M и N');
	readln (M,N);
	while M<>N **do**
	begin
	if M>N **then**
	M:=M-N
	else
	N:=N – M:
	end;
	writeln ('НОД=', M)
	end.

Структура алгоритма — цикл-пока с вложенным ветвлением. Цикл повторяет выполнение, пока значения *M* и *N* не равны друг другу. Ветвление заменяет большее из двух значений на их разность.

А теперь посмотрите на трассировочную таблицу (Табл.16) алгоритма для исходных значений *M*=32, *N*=24.

Таблица 16

Шаг	*Операция*	*M*	*N*	*Условие*
1	ввод *M*	32		
2	ввод *N*		24	
3	*M* не =*N*			32≠24, да
4	*M>N*			32>24, да
5	*M:=M-N*	8		
6	*M не =N*			8≠24, да
7	*M>N*			8>24. нет
8	*N:=N - M*		16	
9	*M не =M*			8≠16, да
10	*M>N*			8>16, нет
11	*N:=N-M*		8	
12	*M не =N*			8≠8, нет
13	вывод *M*	8		
14	конец			

В итоге получился верный результат.

Коротко о главном

Последовательность этапов работы программиста при решении задачи на ЭВМ называется технологической цепочкой. Таких этапов шесть:

1. постановка задачи;
2. математическая формализация;
3. построение алгоритма;
4. составление программы на языке программирования;

5. отладка и тестирование программы;
6. проведение расчетов и анализ полученных результатов.

Любой циклический алгоритм может быть построен с помощью команды «цикл-пока» (цикл с предусловием).

Оператор цикла с предусловием в Паскале имеет вид:

while <логическое выражение> **do** <оператор>

Оператор, составляющий тело цикла, может быть простым или составным**.**

Алгоритм Евклида предназначен для получения наибольшего общего делителя двух натуральных чисел. Структура алгоритма Евклида — цикл с вложенным ветвлением.

Ручная трассировка может использоваться для проверки правильности лишь сравнительно простых алгоритмов. Правильность программ проверяется путем тестирования на компьютере.

4.14. Оператор цикла с постусловием

Так называется оператор который имеет формат:

REPEAT <тело цикла > UNTIL < условие выхода из цикла >.

Здесь REPEAT, UNTIL- кодовые слова (англ.: повторять, до тех пор [,пока не будет выполнено]);

< тело цикла > - произвольная последовательность операторов ТП;

< условие выхода из цикла > - выражение логического типа.

Операторы <тело цикла > выполняются хотя бы один раз, после чего проверяется < условие выхода из цикла >: если его значение есть FALSE, операторы <тело цикла > повторяются, в противном случае оператор REPEAT ... UNTIL завершает свою работу.

Обратите внимание: пара REPEAT ... UNTIL подобна операторным скобкам BEGIN ... END, поэтому перед UNTIL ставить точку с запятой необязательно.

Рассмотрим блок-схему алгоритма с оператором повторения Repeat…Until на примере следующей задачи:

Для x, таких, что 1 < x < M, вычислять и выводить члены ряда: x, x^2, x^3, … ,до тех пор, пока очередной член ряда меньше M и последним вывести первый член ряда превысивший или равный M .

Блок-схема программы приведена на рис. 5.

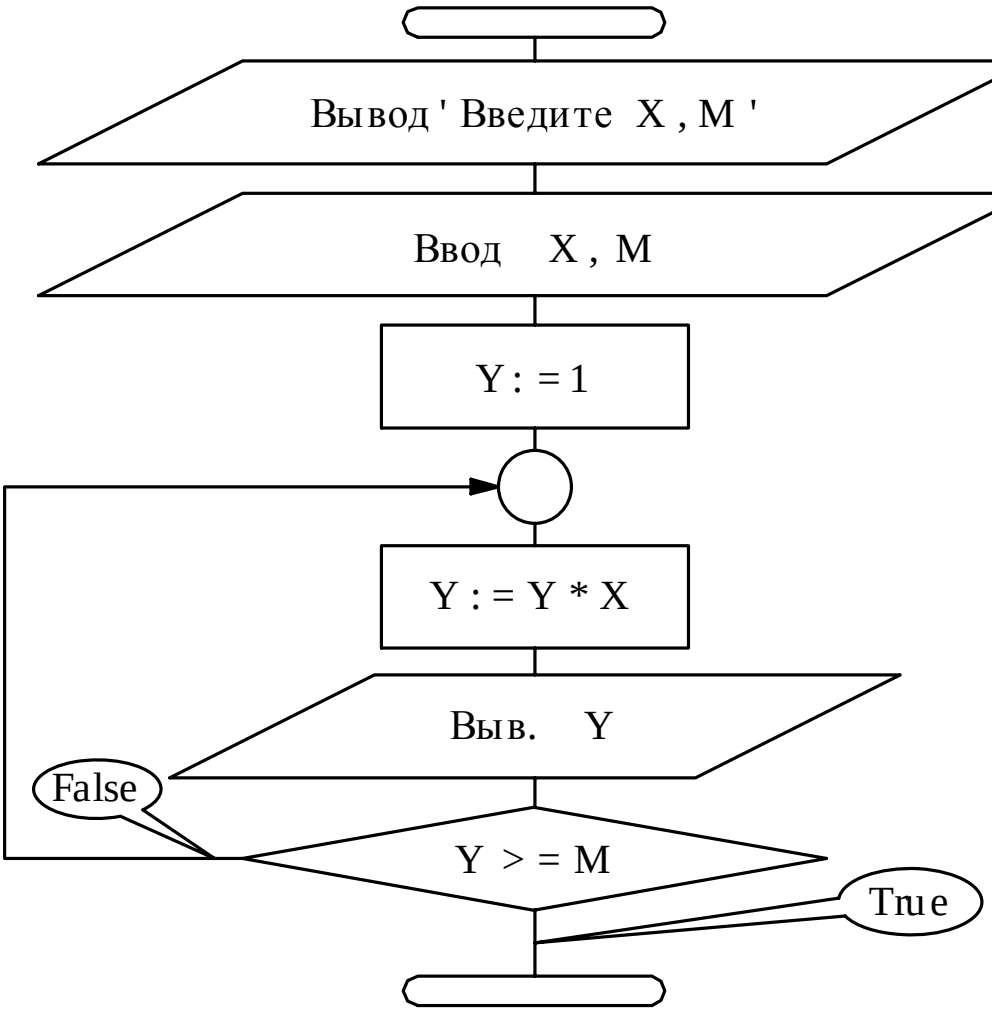


Рис.5 Блок-схема программы

Запишем программу, для приведенной на Рис.5 блок-схемы

```
Program Test_Repeat;
Var
   x, y, M : real;
Begin
```

```
ReadLn(x, M);
y := x;
Repeat
    {В цикле два оператора, но использовать  операторные скобки не надо, т.к. слова – Repeat, Until ограничивают тело цикла.}
    Write ( y );
     y := y * x
Until   y >= M;
End.
```

4.15. Вопросы и упражнения

1. Как блок-схемой и на Алгоритмическом языке представляется команда цикла с предусловием?
2. Как программируется цикл с предусловием на Паскале?
3. Почему алгоритм вычисления *N*! должен быть циклическим?
4. Из каких этапов состоит работа программиста по решению задачи на ЭВМ?
5. Что такое «математическая формализация задачи»?
6. Что такое отладка программы? Что называется тестом?
7. Составить блок-схему алгоритма вычисления суммы всех положительный целых чисел, не превышающих данного натурального числа *N*. Проверить алгоритм трассировкой. Написать программу на Паскале.
8. Дано целое число *X* и натуральное *N*. Составить блок-схему алгоритма вычисления *X* в степени *N*. Проверить алгоритм трассировкой. Написать программу на Паскале.
9. Выполнить на компьютере программу «Evklid» . Протестируйте ее на значениях *M=32, N=24; M=696, N=234*.
10. Составить блок-схему алгоритма и программу нахождения наибольшего общего делителя трех чисел, используя следующую формулу:

НОД(*A,B,C*) = НОД(НОД(*A,B*), *C*).

11. Составить программу нахождения наименьшего общего кратного (НОК) двух чисел, используя формулу: *A* x *B* = НОД(*A, B*) x НОК(*A, B*}

5. Работа с массивами в паскале

5.1. Понятие массива

Массив (array) - это конечный набор элементов одного (базового) типа, которые сохраняются в последовательно размещённых ячейках оперативной памяти и имеют общее имя.

В математике понятию массив соответствуют понятия вектора и матрицы. Различают одно- и многомерные массивы. Двумерный массив данных - это таблица, которая состоит из нескольких строк.

Общий вид конструкции описания типа массив такой:

<имя типа> = **array** [<размер>] **of** <имя базового типа>**;**

Размер (количество элементов) массива чаще всего задают в виде диапазона целых чисел (например **1 . . 100**) или именем некоторого перечислимого типа данных.

Описать массив с помощью введенного имени типа можно в разделе объявления переменных **var**. Имена типов массивов и переменных-массивов указывает пользователь.

5.2. Одномерные массивы.

Пример. Опишем тип массива mas – одномерный массив из 100 дробных чисел. Объявим массивы *A* и *B* типа mas и массив *C* — из 100 элементов-символов.

type mas = **array** [1 **. .** 100] **of real**;

var A, B: mas;

C : **array** [1..100] **of char**;

В этом примере продемонстрированы два способа объявления массивов в программе. Способ, которым введены массивы *A* и *B* является наиболее распространенным.

Над массивами в целом определена единственная команда присваивания. Например, команда *A:= B* все значения элементов массива *B* присваивает элементам массива A (делает значениями элементов массива *A*) при этом массивы должны иметь одинаковый тип.

В большинстве случаев работа ведется с каждым элементом массива отдельно, например: *A* [i] := *B* [j] ; при этом переменные i и j должны получить конкретные числовые значения в предыдущих операторах программы. Для обработки элементов массива как правило используются индексные циклы **for**. Приведем пример простейшей программы, вычисляющей элементы массивов *A и B* и суммирующей элементы массивов *A и B* в массив *C* и выводящей массив *C* на экран.

Program summa;

const n = 10**;**

```
var       A, B, C : array [ 1 . . n ] of integer;
          i : integer;
begin
      for i := 1 to n do
          begin
              A[ i ] := i * i ;
              B [ i ] := 5 * i ;
              C [ i ] := A [ i ] + B [ i  ];
              writeln (C [ i ] :  6); {выводит C[i] в строку консоли и переводит курсор в
начало следующей строки консоли}
           end;
      readln ; {этот оператор без списка ввода будет ожидать нажатия любой клавиши
на клавиатуре,  а человек - оператор в это время может разбираться с результатами, выданными
на экран (консоль). Для возвращения к просмотру текста программы достаточно нажать любую
клавишу}
 end.
```

5.3. Двумерные массивы

Двумерные массивы предназначены для работы с табличными данными. В двумерном массиве элементы определяются именем массива и двумя индексами: первый индекс указывает номер строки, а второй — номер столбца, на пересечении которых находится элемент. Например, *p* [1, 2] - второй элемент первой строки массива p.

Задача. Составить программу для построения таблицы умножения двух чисел (таблицы Пифагора) и занесения её в двумерный массив p. Вывести массив на экран в виде таблицы (матрицы).

```
program Pifagor;
const   n = 9;
type    matr = array [ 1 . . n , 1 . . n ] of integer;
var     p : matr;
        i, j : integer;
 begin
    for i := 1 to n do
      begin
```

for j := 1 **to** n **do** {выводит все элементы i-ой строки матрицы в одну строку консоли}

begin

p [i , j] := i * j;

write (p [i , j] : 6); {выводит p [i , j] в строку консоли }

end;

writeln; {переводит курсор в начало следующей строки консоли для вывода новой строки таблицы умножения}

end;

Readln;

end.

Для доступа к элементам массива необходимо после идентификатора массива *в квадратных скобках указать индекс* нужного элемента или список индексов, определяющий элемент многомерного массива. В качестве индексов могут выступать произвольные выражения, тип которых должен соответствовать типу индексов в описании массива.

Например: *m* [1], *m* [(*I* + 1)*2].

Для двумерного массива доступ к элементам можно записать, например, так:

V[3, 7], *V*[i, j].

Элемент массива считается переменной. Он может получать значения, например, в операторе присваивания, а также участвовать в выражениях.

Для иллюстрации работы с массивами данных составим программу, проверяющую качество работы встроенного генератора псевдослучайных чисел Random. Поскольку этот генератор должен давать последовательность случайных целых чисел, равномерно распределенных на промежутке [0...d) , где d > 0 - параметр обращения к Random, то количество выпадения каждого из чисел: 0,1,2,3, … (d-1) должно быть приблизительно одинаковым для достаточно большой последовательности из N случайных чисел.

Program TestOfRandomGenerator;

const

N = 10000; {количество обращений к Random}

d = 10; {диапазон изменения чисел}

var

measure : array [0..d-l] of word;

i, j : integer;

```
Begin
    for i := 0 to d-1 do
        measure [ i ] := 0;
    for i := 1 to N do
        begin
            j := random(d)
            measure [j] := measure [j] +1 ;
        end;
    for i := 0 to d-1 do
         write(measure [ i ] :5);
    Readln;
End.
```

5.4. Примеры алгоритмов и программ обработки одномерных массивов

Рассмотрим несколько базовых алгоритмов обработки одномерных массивов.

1. Найти сумму значений элементов одномерного массива A=(1.1, -2.2, 6, 4, -5);

Блок-схема алгоритма представлена на Рис. 6.

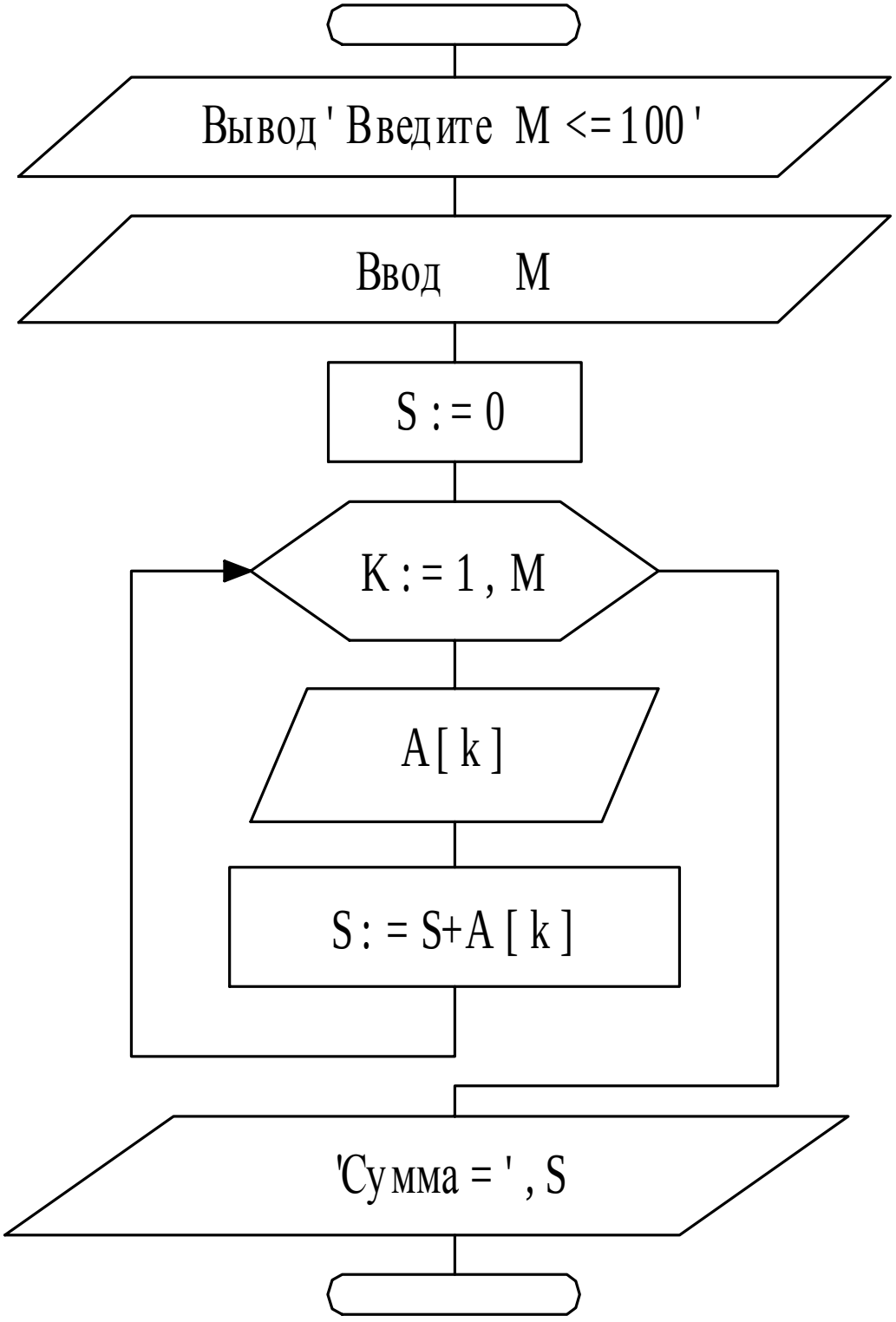


Рис.6 Блок-схема алгоритма

Ниже приведена программа, составленная для рассматриваемого алгоритма.

```
Program Test_Summa;
Var k, M  : integer;       S  : real;
    A  : array[1..100] of real; { тип массива real, т.к. среди заданных значений массива есть числа с десятичной точкой}
Begin

   Write('Введите M <= 100 –количество элементов в вашем массиве');
   ReadLn(M);
   S := 0;
   for k := 1 to M do
      Begin
         Write('Введите очередной элемент вашего массива  ');
          ReadLn(a[k]);{программа считывает введенное число и переводит курсор в новую строку консоли}
           S := S + a[k];
      End;
    WriteLn('Сумма элементов массива A= ', S);

End.
```

2. Найти наибольшее значение элементов одномерного массива A=(1.1, -2.2, 6, 4, -5);

Блок-схема алгоритма представлена на Рис. 7.

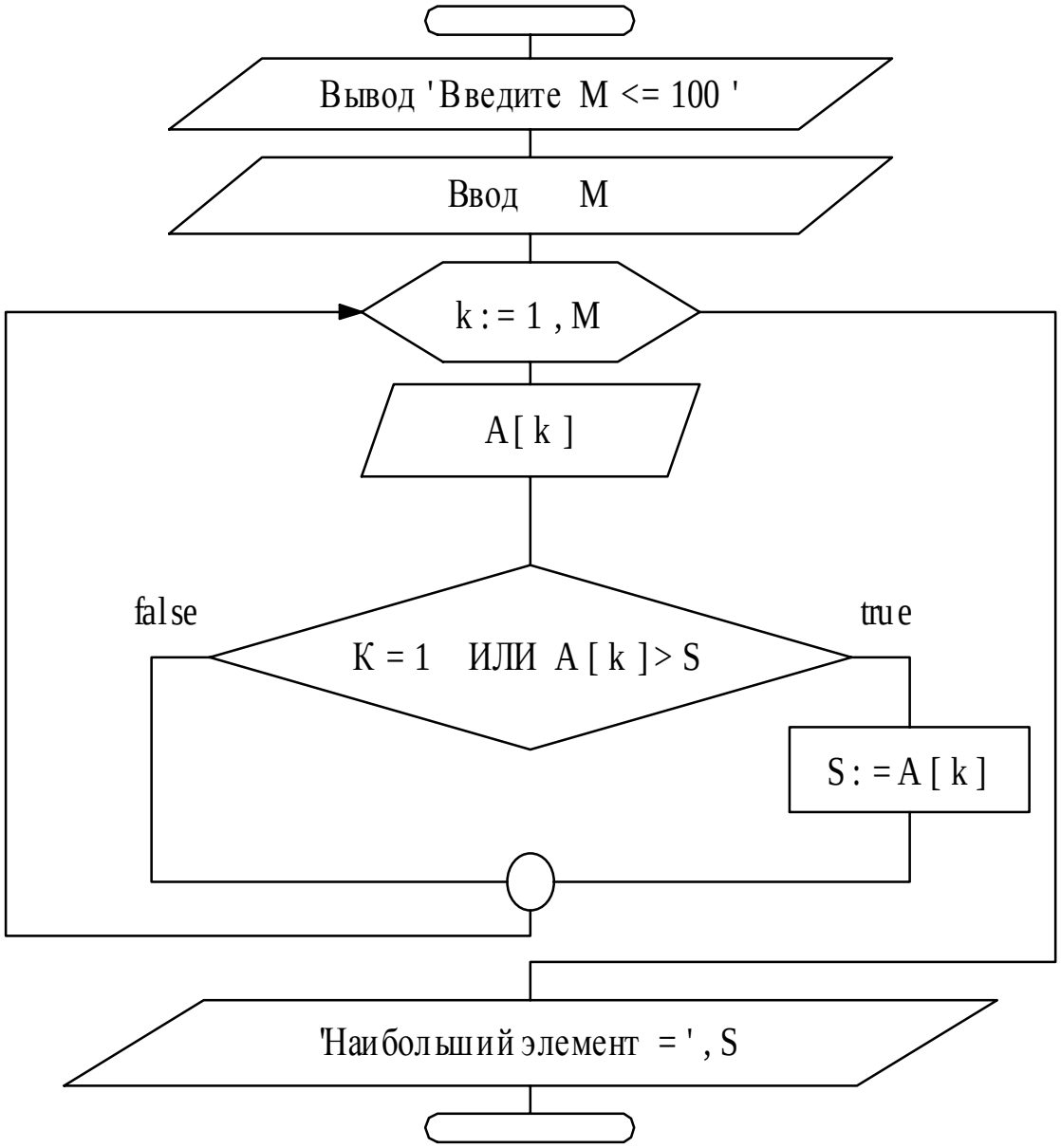


Рис.7 Блок-схема алгоритма

Ниже приведена программа, составленная для рассматриваемого алгоритма.

```
Program Test_Max;
Var     k, M  : integer;{M – количество элементов в обрабатываемом массиве }
        S  : real; {S – переменная для хранения  максимального  элемента массива}
        A  : array[1..100] of  real ;
Begin

   Write('Введите M <= 100 –количество элементов в вашем массиве');
    ReadLn(M);
     S := 0;
     for k := 1 to M do
        Begin
            ReadLn(a[k]);
             if (k = 1) or (a[k] > S) then
                  S := a[k];
         End;
     WriteLn('Наибол. элем. массива A=  ', S);

End.
```

Замечание. Если в условном операторе заменить отношение a [k] > S на a [k] < S, то получится алгоритм определения наименьшего элемента массива.

Два предыдущих алгоритма являются типичными из простейших алгоритмов обработки одномерных массивов.

3. Найти количество положительных элементов массива A=(1.1, -2.2, 6, 4, -5);

Блок-схема алгоритма представлена на Рис. 8

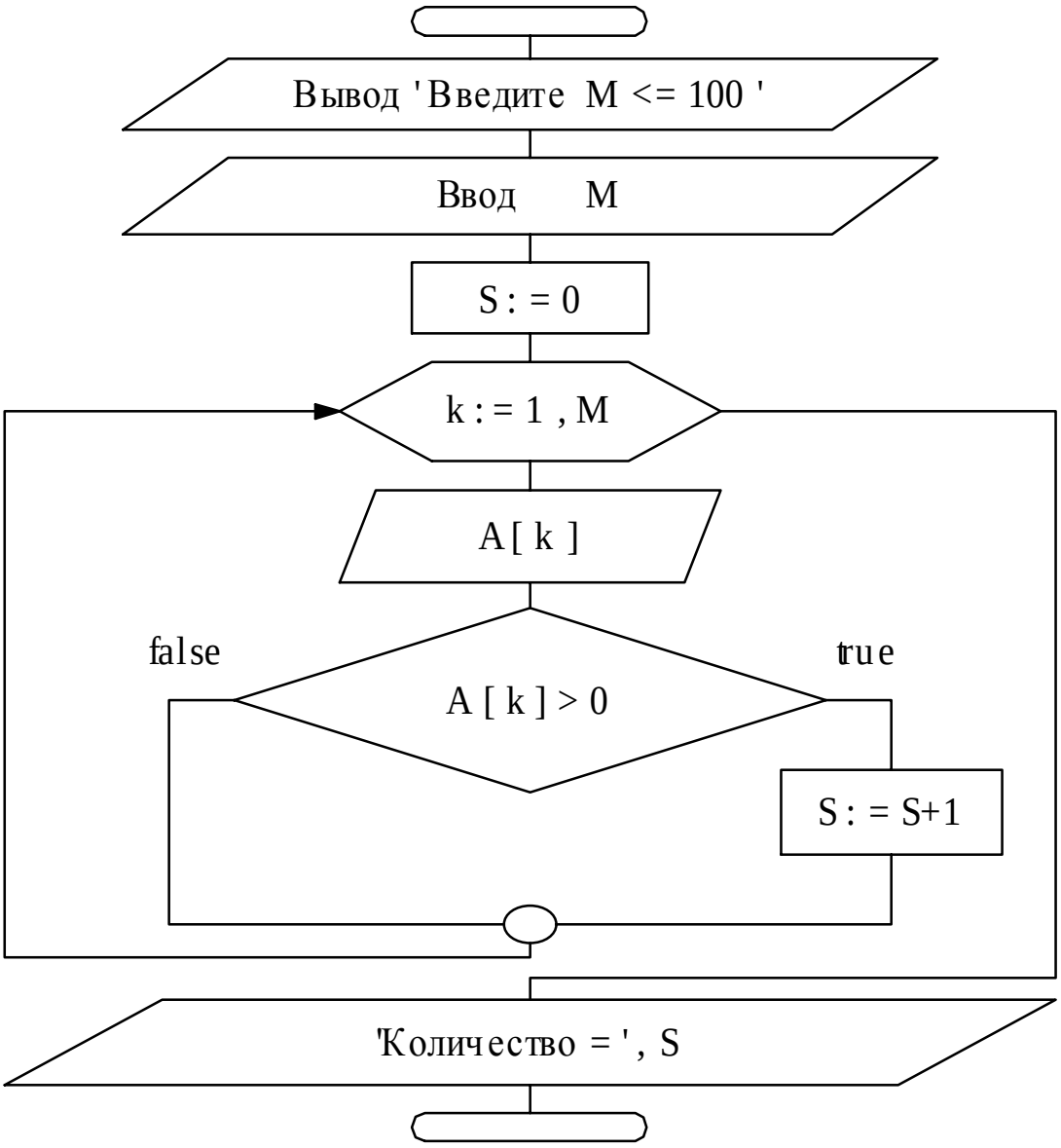


Рис.8 Блок-схема алгоритма

Ниже приведена программа, составленная для рассматриваемого алгоритма.

```
Program Test_Plus;
Var
        k, M  : integer;
        S     : integer; {S – переменная для хранения  количества положительных элементов массива}
        A  : array[1..100] of  real ;
Begin
        Write('Введите M <= 100 –количество элементов в вашем массиве');
        ReadLn(M);
        S := 0;
        for k := 1 to M do
          Begin
              ReadLn(a[k]);
               if  (a[k] > 0 then S := S + 1;
           End;

        WriteLn(‘Кол-во полож. элем.= ', S);
End.
```

Замечание. Если в условном операторе заменить отношение a [k] > 0 на a [k] < 0, то получится алгоритм определения количества отрицательных элементов массива. Если условие заменить на (a [k] > b) and (a [k] <c), то получится алгоритм определения количества элементов, значения которых находятся в интервале от *b* до *c*.

4. Сформировать массив B, записав в него ненулевые элементы массива A. Вывести на строку консоли массив B.

Ниже приведена программа, составленная для рассматриваемого алгоритма.

```
Program Test_Compress;

Var
        k, M, s  : integer; {k-параметр циклов, M –количество элементов в массиве A,
s - количество элементов в массиве B }
        A, B  : array[1..100] of  real ;

Begin
        Write('Введите M <= 100 –количество элементов в вашем массиве');
        ReadLn(M);
        s := 0;
        for k := 1 to M do
          Begin
            Write('Введите очередной элемент вашего  массива ');
            ReadLn(a[k]);
            if  (a[k] > 0) then
              begin
                s := s + 1;
                b [ s ] :=  a [ k ];
              end;
          End;
        Writeln ('количество элементов в новом массиве =  ',  s );
        Writeln (' элементы нового  массива ');
        for k := 1 to s do
          Write( b[k]:6:2, '  '); {Значения элементов нового массива отделяются друг от
друга тремя пробелами}
 End.
```

5.Выполнить сортировку массива A по возрастанию.

Блок-схема алгоритма представлена на Рис. 10.

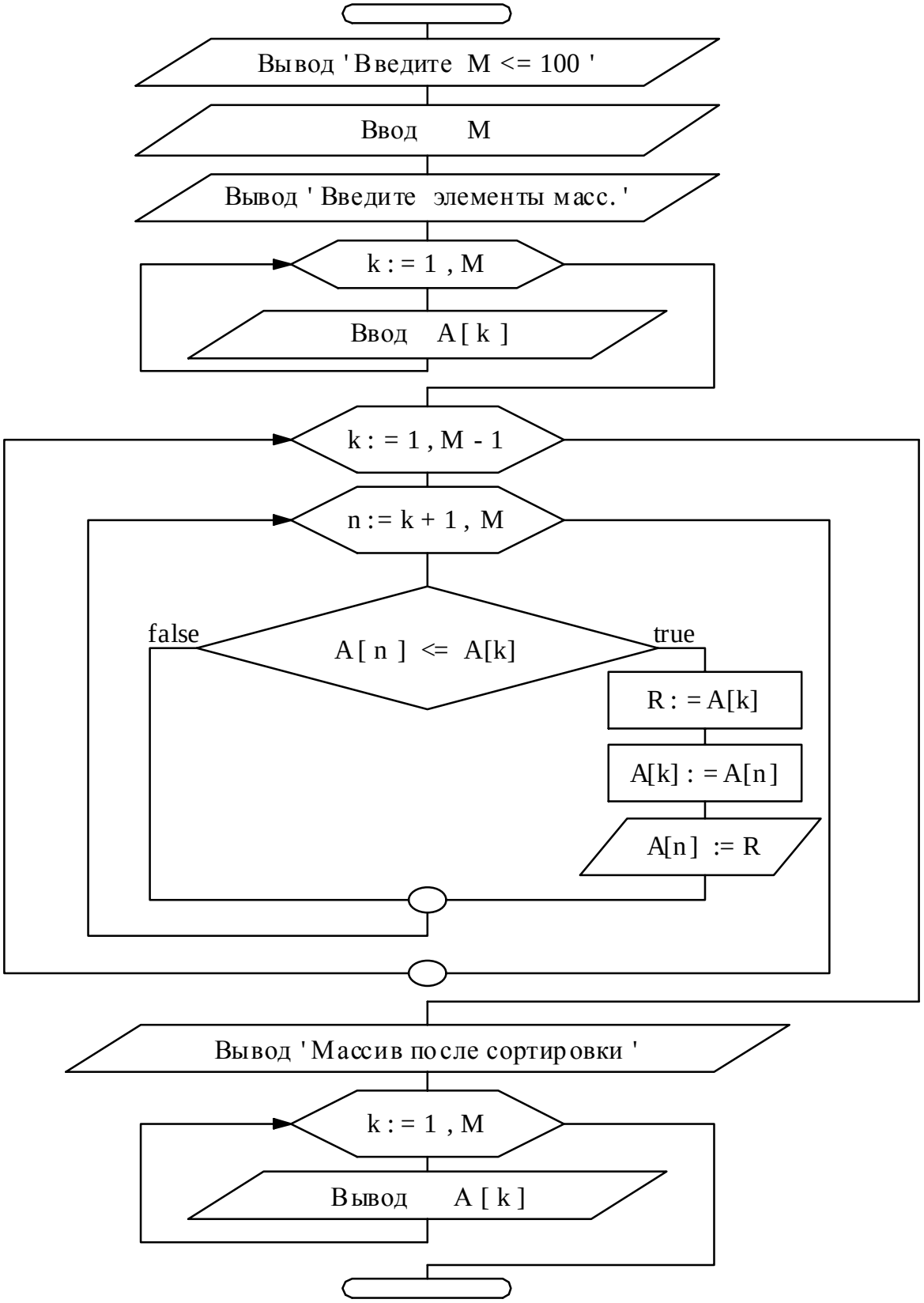


Рис.10 Блок-схема алгоритма

Ниже приведена программа, составленная для рассматриваемого алгоритма.

```
Program Test_Sort;
Var
        k, M, s  : integer
         A, B    : array[1..100] of  real ;
          r   : real;
Begin
        Write('Введите M <= 100 –количество элементов в вашем массиве');
        ReadLn(M);
        Writeln(‘Введите элементы массива ’);
         for k := 1 to M do
           begin
             Write('Введите очередной элемент вашего  массива ');
              ReadLn(a[k]);
           end;
         for k := 1 to M – 1 do       {Фиксируем  К-ую позицию}
            for n := k + 1 to M do    {Перебираем элементы с К+1-го по М-ый }
               Begin            { каждого  n-го сравниваем с К-м}
                 if  (a[ n ] < a[ k ]) then     { меняем их местами}
                    begin
                        r :=  a [ k ];
                        a[ k ] := a [ n ];
                        a [ n ] := r ;
                    end;
                End;
         WriteLn(‘Массив после сортировки по возрастанию’);
         for k := 1 to M do
            Write( a[k]:6:2, ‘     ‘); {Значения элементов отсортированного  массива выводятся в одну строку и отделяются друг от друга тремя пробелами}
End.
```

5.5. Примеры алгоритмов и программ обработки двумерных массивов

Прежде всего, следует заметить, что алгоритмы вычисления суммы, произведения или количества элементов массива, удовлетворяющих некоторому условию, алгоритмы поиска наибольшего и наименьшего значений двумерного массива аналогичны рассмотренным выше алгоритмам для одномерных массивов.

Также обратим ваше внимание на **правильное объявление двумерных статических массивов (матриц=таблиц) в секции объявлений Паскаль программы** на примере конкретной задачи.

Но сначала обсудим еще раз понятие двумерного массива.

Прямоугольную таблицу чисел в математике называют матрицей, в программировании – двумерным массивом. Каждое число массива хранится в отдельной ячейке памяти ЭВМ. Этот набор ячеек удобно представлять себе в виде таблицы:

	1	2	3	…
1				
2				
…				

Вся таблица ячеек имеет имя. Имя массива. Например, A. Каждая ячейка памяти в массиве имеет имя A[i, j], где i – имя ячейки памяти, в которой лежит номер строки, а j – имя ячейки памяти, в которой лежит номер столбца таблицы ячеек A. Указывая в программе имя ячейки массива в левой части оператора присваивания, мы даем команду записать туда число, а если – в правой части, то извлечь из ячейки число и вставить его в формулу вычисления. Строки массива в программе нумеруются, как правило, как в математике от 1. Столбцы тоже. При объявлении имени массива в секции **var** программы, транслятор выделяет таблицу ячеек памяти для хранения чисел – значений элементов массива. Чтобы выбрать последовательно все ячейки памяти массива для обработки в программе используют вложенные циклы. Внешний цикл **for** назначает конкретное число для i – номера строки, а внутренний цикл **for** при фиксированном i перебирает вся значения для j – номера столбца.

В качестве примера, приведем фрагмент программы, который вычисляет сумму элементов массива A, состоящего из 10 строк и 15 столбцов.

```
s:=0; {обнуление сумматора}
for i:=1 to 10 do {заголовок внешнего цикла}
  for j:=1 to 15 do {заголовок внутреннего цикла}
    s:=s+A[i,j]; {тело внутреннего цикла}
```

А теперь приведем и прокомментируем текст программы, которая вычисляет сумму элементов матрицы A, состоящей из mf строк и nf столбцов. Значения для mf и nf вводятся как исходные данные, причем известно, что максимальное возможное значение mf равно 20, а максимальное возможное значение nf равно 30.

Секция описаний переменных и массивов для этой программы и организация ввода значений этих массивов должны стать образцом для ваших программ с двумерными и одномерными массивами.

```
Program P1;

const  mmax = 20; { максимально возможное количество строк в массиве}
       nmax = 30; {максимально возможное количество столбцов в массиве}

type   mas2 = array[1..mmax,1..nmax] of real; {определили имя структуры данных и что
она из себя представляет: количество строк и количество столбцов в ней; количество ячеек
памяти = mmax*nmax, в каждой можно хранить число с дробной частью (real)}

var A : mas2; {транслятор выделяет «таблицу» ячеек памяти для хранения значений
элементов массива A[i,j]}
      i, j : integer; {параметры циклов for}
      mf, nf : integer; {фактическое количество строк и столбцов в массиве (матрице). Эти
два числа программа введет с консоли в режиме диалога с пользователем }
      s : real; {ячейка, в которую программа будет накапливать сумму}
begin
{введем mf и nf}
  Write('Введите количество строк  в вашей матрице  ');
  Readln( mf);
  Write('Введите количество столбцов  в вашей матрице  ');
  Readln( mf);

{Введем значения элементов матрицы (массива)  из строк консоли. В каждую строку
консоли вводим значения элементов одной строки матрицы через пробел. После ввода значения
последнего элемента строки матрицы нажимаем клавишу " Enter"}
  for i:=1 to mf do
    begin
```

```
        for j:=1 to nf do
          Read (A[i,j]);

        Readln; {Чтение кода конца строки. Переход к новой строке ввода}

    end;

  s:=0;
  for i:=1 to mf do
     for j:=1 to nf do
        s:=s+A[i,j];

   Writeln('Сумма элементов массива A=  ', s);

end;
```

Приведем еще три типичных алгоритма и программы работы с матрицами. Внимательно их изучите. Но заметьте, что для упрощения, ввод данных в этих программах сделан не так как в предыдущей программе (без mf и nf). Он хуже. Объясните почему.

1. Рассмотрим алгоритм транспонирования матрицы. *Задана матрица A*. *Определить элементы матрицы* $B = A^T$. *По определению элемент матрицы B определяется формулой* $b_{k,r} = a_{r,k}$ (1-ая строка *A* становится 1-ым столбцом *B*, 2-ая строка *A* – 2-ым стобцом *B* и так далее)

На Рис.11 изображена блок-схема алгоритма.

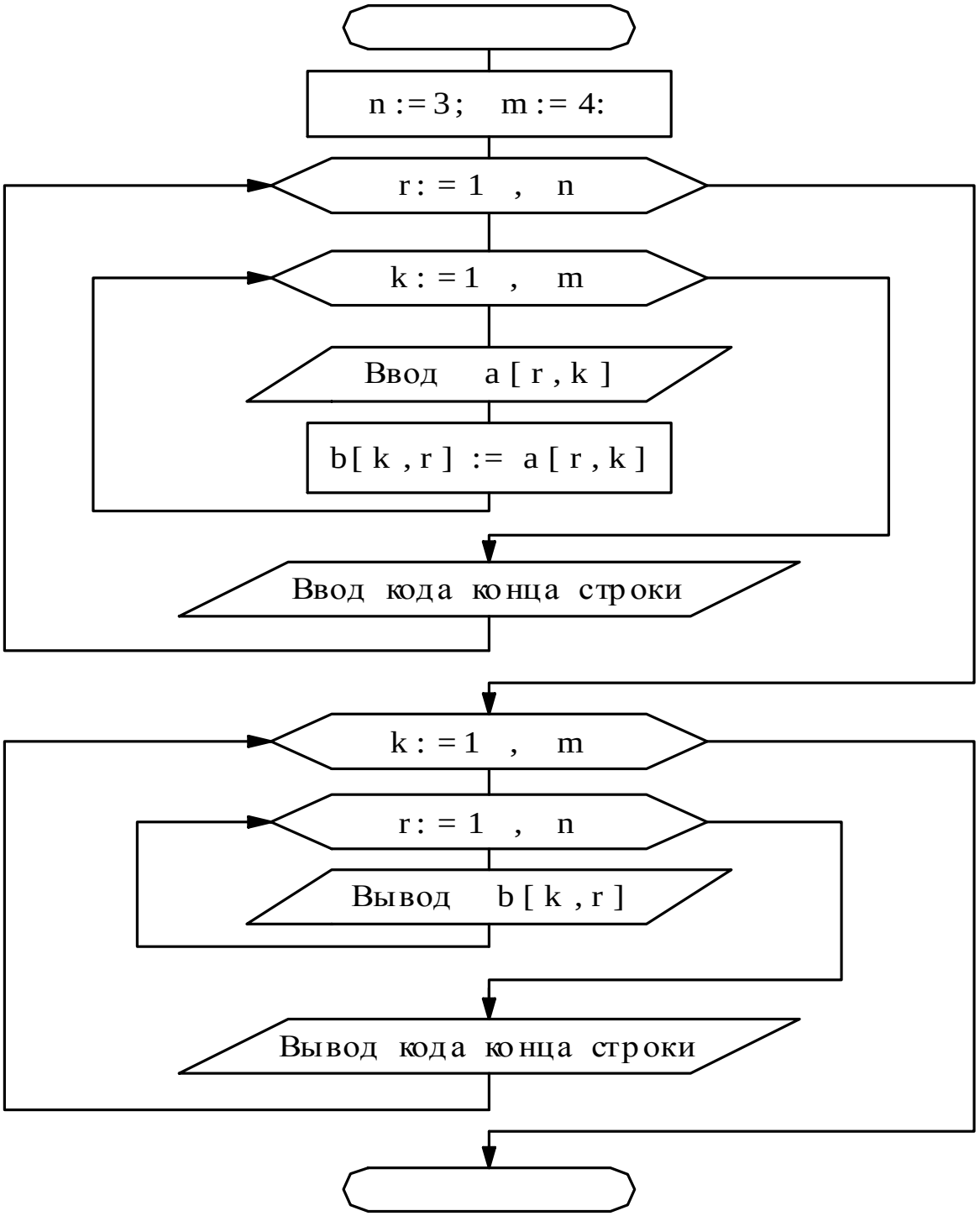


Рис.11 Блок-схема алгоритма

Запишем по блок-схеме программу.

```
Program Trans_Matrix;
Const   n = 3;
        m = 4;
Type  matr_nxm = array[1..n,1..m] of real; {}
      matr_mxn = array[1..m, 1..n] of real; {}
Var     A : matr_nxm;
        B : matr_mxn;
        r, k : integer;
Begin
  Writeln('Введите элементы двумерного массива (матрицы = таблицы) из ');
  Writeln('трех строк и четырех столбцов. Элементы строки вводите в строку консоли через пробел.');
  Writeln(После ввода последнего элемента строки ');
  Writeln('  нажмите клавишу «Ввод» = «Enter»');
 for  r :=  1 to n do
   begin
     for k := 1 to m do {читает  r-ую  строку матрицы из одной строки консоли}
       begin
         Read(a[r, k]);
         b[k, r] := a[r, k];
       end;
     Readln; {Чтение кода конца строки. Переход к новой строке ввода}
   end;
 for k :=  1 to m do
   begin
     for r := 1 to n do   {Цикл выводит строку матрицы на одну строку экрана}
         Write(b[k, r]);
     writeln;  {Переход к новой строке вывода}

   end;
End.
```

2. Рассмотрим алгоритм сложения двух матриц. *Даны матрицы A и B. Определить матрицу C = A + B.*

Правило сложения матриц определяется формулой $c_{r,k} = a_{r,k} + b_{r,k.}$, где r и k – соответственно номер строки и столбца в каждой из матриц (сложение элементов матриц, стоящих в одинаковых позициях)

На Рис.12 представлена блок-схема алгоритма.

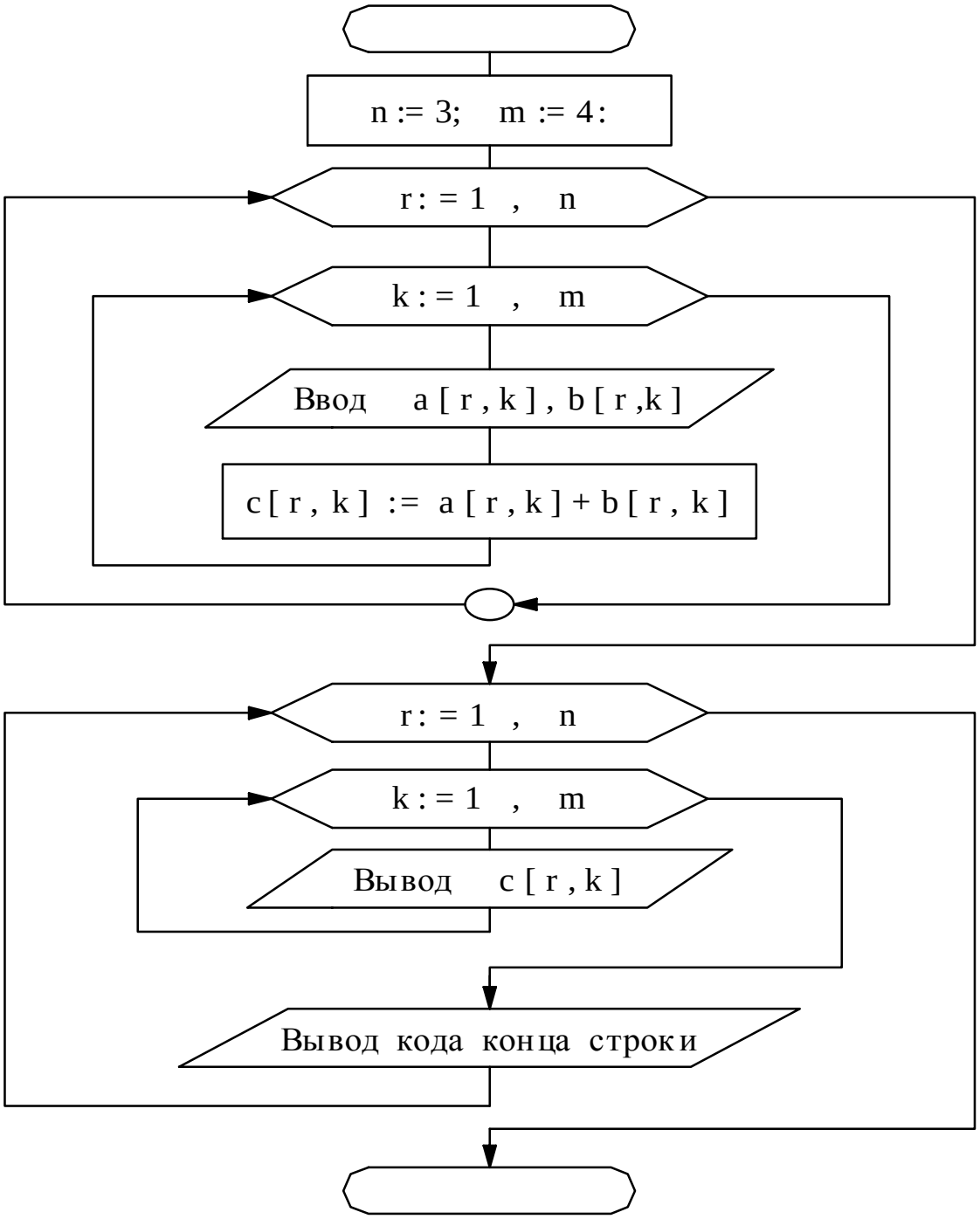


Рис.12 Блок-схема алгоритма

Запишем по блок-схеме программу.

```
Program Add_Matrix;
Const   n = 3;
        m = 4;
Type
        matr_nxm = array[1..n,1..m] of real;
Var
        A, B, C : matr_nxm;
        r, k : integer;
Begin

      for r :=  1 to n do
        for k := 1 to m do
          begin
            Readln(a[r, k], b[r, k]);
            c[r, k] := a[r, k] + b[r,k];
          end;

      for r :=  1 to m do
        begin
          for k := 1 to n do
            Writeln(c[r, k]);

          writeln;
        end;
End.
```

3. Рассмотрим алгоритм произведения двух матриц (Рис.13).

*Заданы матрицы A и B .Определить матрицу C = A * B. Элемент матрицы C стоящий в k-ой строке и в r-ом столбце вычисляется как скалярное произведение k-ой строки матрицы A(как вектора) на r-ый столбец матрицы B (как вектор).*

Чтобы не усложнять понимание алгоритма, укажем только ту часть алгоритма, где выполняется перемножение матриц. Ввод исходных матриц и вывод матрицы результата обозначим кратко.

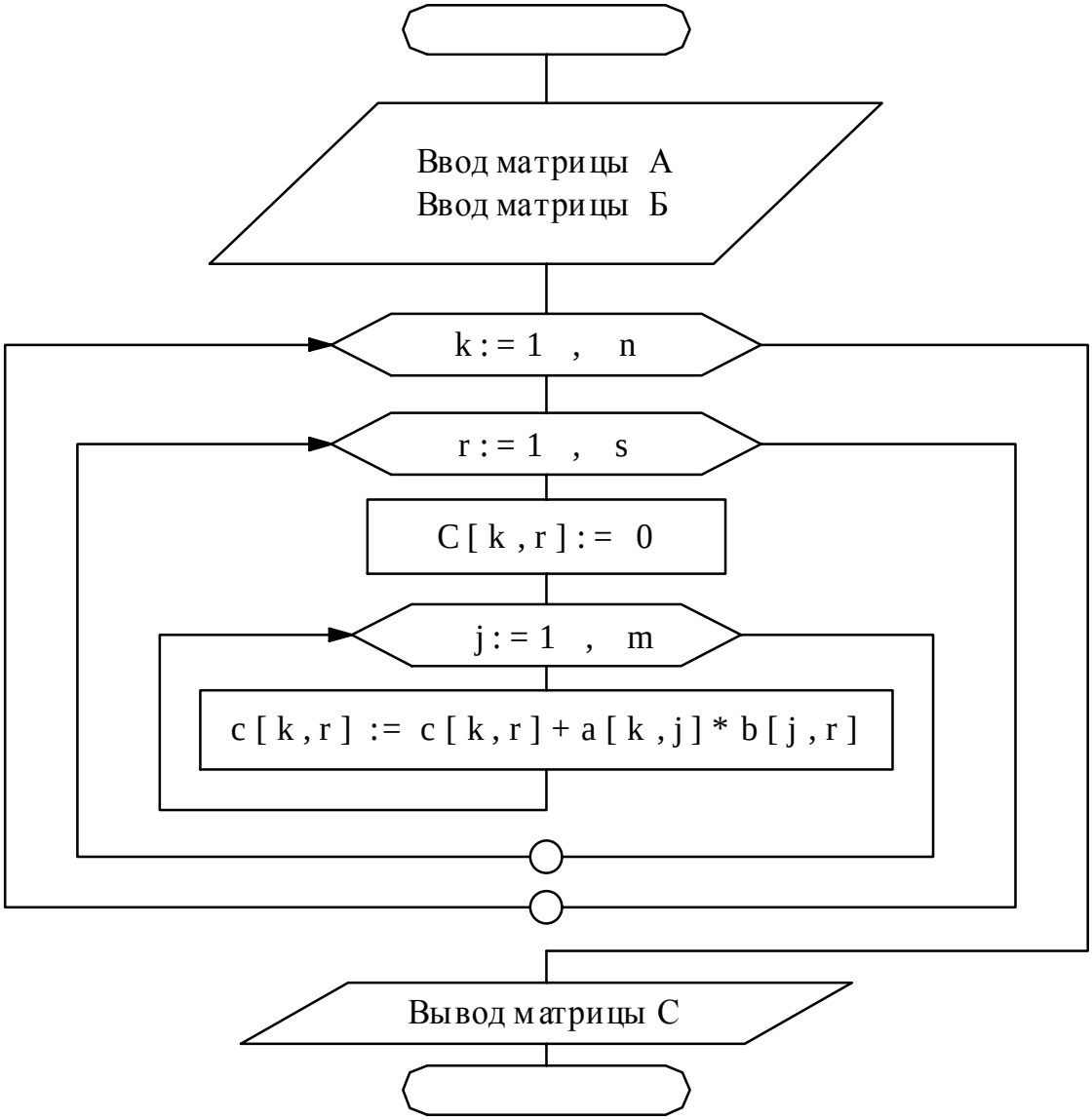


Рис.13 Блок-схема алгоритма

Программа имеет вид.

```
Program Mult_Matrix;
Const    n = 3;
         m = 4;
          s = 2;
Type
        matr_nxm = array1..n,1..m] of real;
        matr_mxs = array[1..m,1..s] of real;
        matr_nxs = array[1..n,1..s] of real;
Var
        A : matr_nxm;
        B : matr_mxs;
        C : matr_nxs;
        r, k, j : integer;
Begin
{Здесь вводятся исходные матрицы A и B}

for  k :=  1 to n do
   for  r :=   1 to s do
      begin
         c[k, r] := 0;
         for  j := 1 to m do
              c[k, r] := c[k, r] + a[k, j]*b[j, r];
      end;

{Здесь выводится результирующая матрица C}
End.
```

6. Работа со строковыми переменными

6.1. Строковые переменные

Строковый тип данных определяет множество символьных цепочек произвольной длины (от нуля символов до заданного их числа). Для определения строкового типа используется зарезервированное слово STRING, вслед за которым в квадратных скобках может быть указана максимальная длина строки.

```
Type
    Line = string [80];
Var
    MyLine : Line;
```

В приведенном примере переменная MyLine в качестве своего значения может иметь любую последовательность символов произвольной длины в пределах от 0 до 80 символов.

Указание максимальной длины для строки может быть опущено, в этом случае подразумевается число 255. Такая длина является максимально возможной для строковых типов.

6.2. Операции со строковыми переменными

Значения строковой переменной может быть присвоено оператором присваивания или введено оператором ввода.

Для строк определена операция конкатенации («+»), например:

```
MyLine := ‘Строка’;
MyLine := MyLine + ‘стала длиннее’
```

Значением переменной MyLine будет строка ‘Строка стала длиннее’

Важнейшее отличие строк от обычных символьных массивов заключается в том, что строки могут динамически изменять свою длину.

В случае присваивания строковой переменной строкового выражения с длиной, большей, чем максимально допустимая для данной переменной, происходит отбрасывание последних символов выражения до максимальной длины строки.

Для описания строковой переменной длиной N отводится N+1 байтов памяти, из которых N байтов предназначены для хранения символов строки, а один байт – для значения текущей длины этой строки. Для определения текущей длины строки, используется стандартная функция **LENGTH**, единственным параметром которой является выражение строкового типа. Эта функция возвращает целое значение, равное длине строки.

Кроме операции конкатенации, над значениями строковых типов определены операции сравнения: <, <=, >, >=, =, <>.

Строки сравнивают по первой паре несовпадающих символов. Чей символ меньше (раньше в кодировочной таблице (в алфавите)) та строка меньше. Если у двух строк имеющиеся пары символов совпадают, но одна строка короче другой, то она считается меньше.

Приведем примеры:

‘*abce*’ < ‘*abcx*’ ,так как у первой пары несовпадающих символов *e* < *x* (стоит в алфавите раньше);

‘*abce*’ > ‘*abc*’, так как имеющиеся пары символов совпадают, но 2-ая строка короче.

Элементы строки нумеруются целыми числами, начиная с единицы. Доступ к отдельным элементам строк производится аналогично доступу к элементам одномерного массива: после имени строковой переменной необходимо в квадратных скобках указать выражение целого типа, обозначающее номер элемента строки. Ниже приведенная программа формирует строку из заглавных букв латинского алфавита.

```
Var   Str : string[26];  I  :  integer;
Begin
   Str :='';
   For I := 1 to 26 do
        Str := Str + Chr (Ord ('A') + I – 1);
   WriteLn(Str);
End.
```

Поясним программу. Встроенная функция **Ord** (‘A’) возвращает код(номер) символа *A* в кодировочной таблице (в алфавите ЭВМ). Номер каждой последующей буквы в кодировочной таблице (алфавите ЭВМ) на 1 больше. Встроенная функция **Chr** (**Ord** (‘A’) + I – 1) по вычисленному для неё номеру возвращает соответствующий символ алфавита, который заносится очередным символом в строку Str.

Кроме стандартной функции **LENGTH**, в ТП имеется несколько процедур и функций, работающих со строками.

1. **Concat** (s1 [,s2, …, sn] : string) : string;

Эта функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина результирующей строки превышает 255 символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации и работает немного эффективнее, чем эта операция.

2. **Copy** (s : string; Index : integer; Count : integer) : string;

Функция возвращает подстроку, выделенную из исходной строки S, длиной Count символов, начиная с символа с номером Index.

3. **Pos** (Substr, s : string) : byte

Функция производит поиск в строке S подстроки Substr. Результатом функции является номер позиции подстроки в исходной строке. Если подстрока не найдена функция возвращает значение 0.

4. **Delete** (Var s : string; Index, Count : integer);

Процедура производит удаление из строки-параметра S подстроки длиной Count символов, начиная с символа с номером Index.

5. **Insert** (Source : string; Var S : string; Index : integer);

Процедура предназначена для вставки строки Source в строку S, начиная с символа с номером Index.

6.3. Примеры алгоритмов и программ обработки строк

При составлении программы обработки строки строку рассматривают как одномерный массив символов. Приведем несколько программ обработки текста. В качестве полезного упражнения рекомендуем по каждой программе составить блок-схему алгоритма. В программах для простоты будем считать, что длина вводимого текста не превосходит 255 символов и весь текст можно ввести в одну строковую переменную.

Задача 1. *Удалить все начальные и все конечные пробелы в тексте, а также все лишние пробелы между словами текста. Между словами должно быть не более одного пробела.*

Текст, полученный после такой обработки, назовем стандартным. Программа имеет вид

```
Program Standart_Str;
Var  S : string;
       i : integer;
          Begin
              Writeln (‘ Введите текст ‘)
               Readln ( S );
               While S[ 1 ] = ‘ ‘  do       { Пока S[1] = пробелу}
                    Delete ( s , 1 ,1 );       { В цикле избавились от начальных пробелов}
               While S [ Length ( S )] = ‘ ‘  do       { Пока S[ последний] = пробелу}
                    Delete  (  S  ,  Length  (  S  ) ,1  );       {В цикле избавились от концевых пробелов}
```

```
        i := 1;
        While  i <  Length ( S )  do
            Begin
                While ( S[ i ] = ‘ ‘ ) and ( S[ i +1] = ‘ ‘ ) do
                    Delete ( S , i+1, 1 ); { В этом вложенном цикле среди найденных нескольких, стоящих рядом пробелов, удалили все кроме первого}
                    i := i + 1;  {Перешли в следующую позицию, для поиска следующего набора из двух или более пробелов}
                End;
            Writeln (  S  );
    End.
```

Задача 2. *В стандартном тексте подсчитать количество слов*.

Нетрудно сообразить, что количество слов будет на 1 больше количества разделяющих их пробелов. Подсчетом пробелов программа и займется.

```
Program Kol_ slov;
Var  S : string;
     i , l , k : integer;
   Begin
      Writeln (‘ Введите стандартный текст ‘)
      Readln ( S );
      l := Length ( S ) ;
      k := 0;
      for  i := 1 to l  do
          If  S[ i ] = ‘ ‘ then
              k := k + 1;

      k := k + 1;
      Writeln ( ‘ текст состоит из  ‘ , K , ‘ Слов ‘ );
   End .
```

Задача 3. Программа должна определить является ли введенное слово полиндромом. Полиндром – слово, которое читается одинаково с начала до конца и с конца до начала. Для простоты будем считать, что слово вводится либо только строчными либо только заглавными буквами. Например: казак, АЛЛА.

```
Program Polindrom;
Var S : string;
    i , m , k : integer;
      Begin
        Writeln (‘ Введите слово ‘)
         Readln ( S );
         m := Length ( S ) div 2 ; {Количество пар символов, равноудаленных от начала и конца слова}
         k := 1;  {Предполагаем, что слово – полиндром}
         for  i := 1 to m  do
           if  S [ i ]  < >  S [Length ( S ) + 1 - i ]  then
              k := 0 ;{Если найдется хотя бы одна пара не равных равноудаленных от начала и конца слова символов, то k будет равна 0}
         if  k = 1  then
           Writeln ( ‘ Слово - полиндром ‘ );
         else
           Writeln ( ‘ Слово – не полиндром ‘ );
      End.
```

Задача 4. Определить количество разных вхождений заданной подстроки в заданную строку. Два вхождения разные, если они не имеют общих символов.

```
Program Kol_SubS_v_Str;
Var  S , Sdubl , SubS: string;
     k , x : integer;
        Begin
           Writeln (‘ Введите текст ‘)
           Readln ( S );
           Writeln (‘ Введите подстроку ‘)
           Readln ( SubS );
            Sdubl := S;  {Делаем копию S}
            k := 0 ;  { Обнуление счетчика вхождений}
             x := Pos( SubS, Sdubl );
            While  x > 0  do
                Begin
                    k := k + 1;  {Раз нашли вхождение увеличиваем счетчик на 1}
                     Delete ( Sdubl , 1 , x + Length(SubS) - 1 ); { Удалили начало строки по последний символ найденного вхождения включительно}
                     x := Pos( SubS, Sdubl );{ Поиск следующего вхождения в урезанной строке}

                End;
              Writeln (  ‘ Количество найденных вхождений = ’ , k );
        End.
```

Рекомендуемая литература

1. Информатика: алгоритмизация и программирование. Учеб. пособие / В. В. Гаврилова, А. А. Кудлаев, А. М. Сёмов ;-М. : МИИГАиК, 2010.-82 с.: ил.

2. Информатика. Технология программирования. Алгоритмы обработки массивов. Учеб. пособие / А. А. Кудлаев;-М. :МИИГАиК, 2019.-44 с.: ил

3. Информатика. Базовый курс. 2-е издание / Под ред. С.В. Симоновича. – СПб.: Питер, 2009.-640с.: ил.

4. Князева М.Д. Алгоритмика: от алгоритма к программе. Учебное пособие -М.: КУДИЦ-ОБРАЗ, 2006.-192 с.: ил.

5. Могилев А.В. Информатика: учеб. пособие для студ. пед. вузов / А.В. Могилев, Н.И. Пак, Е.К. Хеннер ; под ред. Е.К. Хеннера. – 6-е изд., стер. – М.: Издательский центр «Академия», 2008.-848 с.: ил

Содержание