**Тема 8. ОСНОВЫ АЛГОРИТМИЗАЦИИ**

1. **Понятие алгоритма. Свойства алгоритма. Способы описания алгоритма.**
2. **Линейная алгоритмическая конструкция**
3. **Комбинированные алгоритмы**
4. **Понятие алгоритма. Свойства алгоритма. Способы описания алгоритма.**

Уровень развития информационных технологий требует от любого специалиста подготовки не только по собственному профилю, но и знаний основ алгоритмизации и программирования. Основу деятельности специалиста практически любой области составляет умение ставить задачи, разрабатывать алгоритмы, получать решения, производить анализ полученных данных и делать выводы.

Алгоритмизация, алгоритмика – это раздел информатики, который занимается изучением:

* методов и приемов управления логическими и вычислительными устройствами;
* процесса планирования последовательности шагов решения задачи (создания алгоритма), результатом которого является выделение этапов процесса обработки данных, формализованная запись содержания этих этапов и определение порядка их выполнения;
* методов, приемов построения и свойств алгоритмов.



***Алгоритмизация****– это представление неформального, неточного и неполного описания известного метода решения задачи в виде четкого алгоритма.*

Понятие алгоритма такое же основополагающее для информатики, как и понятие информации.

***Алгоритм****– описанная на некотором языке точная конечная система правил, определяющая содержание и порядок действий над некоторыми объектами, строгое выполнение которых дает решение поставленной задачи.*

 Или (более коротко):

***Алгоритм****– это строго определенная последовательность действий, необходимых для решения данной задачи.*

 Это не математическое, а логическое определение алгоритма, дающее его содержательно-логическую суть.

***Разработать******алгоритм решения****означает разбить задачу на последовательно выполняемые этапы.*

Можно сказать, что алгоритм описывает процесс преобразования исходных данных в результаты, т. к. для решения любой задачи необходимо:

* ввести исходные данные;
* преобразовать исходные данные в результаты (выходные данные);
* вывести результаты.

***Исполнитель алгоритма****– некоторая абстрактная или реальная (техническая, биологическая или биотехническая) система, которая понимает команды алгоритма и умеет правильно их выполнять.*

Алгоритм предназначен всегда для определенного исполнителя – человека, робота, компьютера, языка программирования и т. д.

***Выполнить алгоритм****–значит применить его к решению конкретной задачи, т. е. выполнить запланированные действия по отношению к определенным исходным (входным) данным для получения искомого результата (выходные данные).*

В процессе работы исполнитель может создавать и использовать данные, не являющиеся выходными (промежуточные данные), но необходимые для перехода от входных данных к выходным (рис. 1).

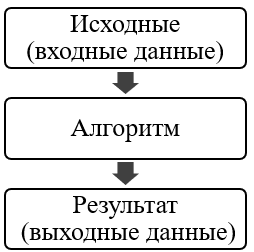


Рис. 1

**Исполнителя характеризуют:**

* среда
* элементарные действия
* система команд
* отказы

Умение выполнять определенные команды является с***войством***, характеризующим любого исполнителя.

***Системой команд исполнителя****называется совокупность всех команд, которые данный исполнитель может выполнять*

 Алгоритм описывается в командах определенного исполнителя, который будет его реализовывать.

***Среду исполнителя****образуют объекты, над которыми исполнитель может совершать действия*

Исходные данные и результаты любого алгоритма всегда принадлежат среде того исполнителя, для которого предназначен алгоритм. Для каждой команды должны быть заданы условия применимости (в каких состояниях среды может быть выполнена команда) и описаны результаты выполнения команды. После вызова команды исполнитель совершает соответствующее *элементарное действие*. *Отказы* исполнителя возникают, если команда вызывается при недопустимом для нее состоянии среды.

***Разработка алгоритма решения задачи****- это разбиение задачи на последовательно выполняемые этапы.*

Результаты выполнения предыдущих этапов могут использоваться при выполнении последующих. Содержание каждого этапа и порядок выполнения этапов должны быть четко указаны. Отдельный этап алгоритма должен быть простым и понятным без пояснений, либо представлять собой другую, более простую задачу, алгоритм решения которой известен (разработан заранее).

**Описание алгоритма решения задачи выполняется в соответствии со следующими правилами:**

1. *определить исходные данные задачи;*
2. *разбить процесс решения задачи на этапы, понятные и однозначные для исполнителя;*
3. *указать порядок, в котором выполняются этапы, а также признак завершения процесса;*
4. *определить, что является результатом решения задачи.*

**Свойства алгоритма**

**Свойства алгоритма**

Понятие «алгоритм» имеет схожий смысл с такими понятиями, как «набор инструкций», «последовательность действий», «метод». Однако, в отличие от них, алгоритм характеризуется определенными свойствами.

***Свойства алгоритма****– это набор свойств, отличающих алгоритм от любых предписаний и обеспечивающих его автоматическое исполнение*

 Сущность алгоритмизации вычислительного процесса проявляется в следующих действиях, отражающих его свойства:

* выделении законченных частей вычислительного процесса;
* формальной записи каждого из них;
* назначении определенного порядка выполнения выделенных частей;
* проверки правильности выбранного алгоритма по реализации заданного метода вычислений.

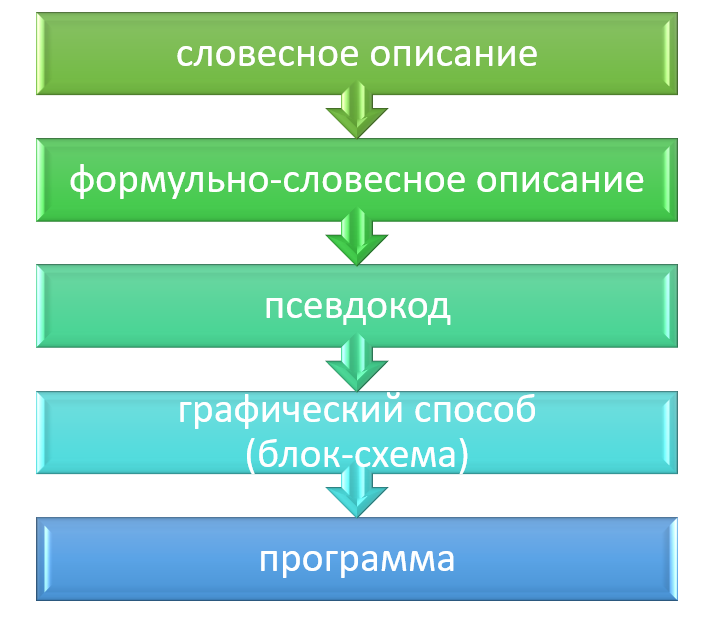


1. ***Дискретность***(разрывность). Свойство, характеризующее его структуру: каждый алгоритм состоит из отдельных законченных действий, т. е. «делится на шаги». Каждый шаг называется командой (инструкцией). Только после завершения одной команды можно перейти к выполнению следующей.
2. ***Массовость.*** Свойство, характеризующее применимость алгоритма ко всем задачам рассматриваемого типа, при любых исходных данных. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.
3. ***Формальность.*** Свойство, указывающее на то, что любой исполнитель, способный воспринимать и выполнять инструкции алгоритма, действует формально, т. е. отвлекается от содержания поставленной задачи и лишь строго выполняет инструкции. Другими словами, механически выполняя все указанные в алгоритме этапы в требуемом порядке, исполнитель может всегда правильно решить задачу.
4. ***Результативность.*** Свойство, состоящее в том, что любой алгоритм должен завершаться за конечное (пусть даже очень большое) число шагов: либо приводить к решению задачи, либо останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.
5. ***Определенность***(детерминированность, точность). Свойство алгоритма, указывающее на то, что каждый шаг алгоритма должен быть строго определен и не допускать различных толкований; также строго должен быть определен порядок выполнения отдельных шагов.

**Способы описания алгоритма**

На любой стадии существования алгоритмы представляют с помощью конкретных изобразительных средств, состав и правила употребления которых образуют конкретные способы или формы записи.

**К настоящему времени сложились пять наиболее употребительных способов записи:**



**1. Словесное описание**

Представляет собой алгоритм – инструкцию о выполнении действий в определенной последовательности с помощью слов и предложений естественного языка.

Форма изложения произвольна и устанавливается разработчиком.

Этот способ описания не имеет широкого распространения, т. к. строго не формализуем (под «формальным» понимается то, что описание абсолютно полное и учитывает все возможные ситуации, которые могут возникнуть в ходе решения), допускает неоднозначность толкования при описании некоторых действий, страдает многословностью.

***Пример 1.****Составить алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел.*

Словесный алгоритм:

1. Задать два числа.
2. Если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма.
3. Определить большее из чисел.
4. Заменить большее из чисел разностью большего и меньшего из чисел.
5. Повторить алгоритм с п. 2.

**Пример 2*.***Составить алгоритм выполнения домашнего задания по математике.

Словесный алгоритм:

1. Открыть дневник.
2. Посмотреть, что задано.
3. Взять учебник и тетрадь по предмету.
4. Прочитать параграф и выполнить письменные упражнения.
5. Сравнить полученные результаты с ответами в конце учебника.
6. Если ответы совпадают, закрыть тетрадь и учебник, если нет – повторить алгоритм с п. 4.

**2. Формульно-словесный способ**

Содержит формальные символы и выражения (формулы) в сочетании со словесными пояснениями. Т. е. алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Этот способ описания нагляден, лаконичен, но не является строго формальным.

***Пример 3.****Составить алгоритм вычисления следующего выражения: у = 2а – (х + 6)*

Формульно-словесный алгоритм:

1. Ввести значения ***а*** и ***х***
2. Найти сумму **(*х* + 6)**
3. Найти произведение **(2\**а*)**
4. Вычислить ***y*** как разность***y*= 2*a –* (*x*+ 6)**
5. Вывести ***у*** как результат вычисления выражения

**3. Псевдокод**

Представляет собой описание структуры алгоритма на естественном, частично-формализованном языке, позволяющее выявить основные этапы решения задачи перед точной его записью на языке программирования.

В псевдокоде используются некоторые формальные конструкции и общепринятая математическая символика. Данный способ тесно связан со структурным подходом к программированию. Псевдокод занимает промежуточное положение между естественным языком и языком программирования. Его применяют преимущественно для того, чтобы подробнее объяснить работу программы, что облегчает проверку правильности программы.

***Пример 5****. Составить алгоритм сложения двух чисел.*

Псевдокод:

1. Ввод двух чисел *a* и *b*.
2. Вычисление суммы *S* = *a* + *b*.
3. Вывод *S*.
4. Конец.

***Пример 6.****Составить алгоритм заполнения зачетной ведомости группы из 20 студентов (i – номер студента)*

Псевдокод:

1. начало цикла  
   **для**  
   *i*от 1 до 20 с шагом 1  
   **повторять**:  
   1.1.    ввести фамилию студента  
   1.2.  поставить оценку  
   **конец цикла (кц)**
2. вывод ведомости

**4. Программа**

Алгоритм, записанный в виде последовательности команд, понятных ЭВМ (машинных команд). Для записи алгоритмов в виде программ характерна высокая степень формализации. Перед составлением программ чаще всего используются словесно-формульный или графический способы.

***Пример 7.****Составить алгоритм сложения двух чисел на языке Python*

Программа:

          num1 = input('Введи первое число: ')

          num2 = input('Введи второе число: ')

          sum = float(num1) + float(num2)

          print('Результат сложения двух чисел равен:', sum)

**5. Графическая запись** (**блок-схема)**

Описание структуры алгоритма с помощью геометрических фигур с линиями связями, показывающими порядок выполнения отдельных инструкций. Описание алгоритмов с помощью схем – один из наиболее наглядных и компактных способов.

Этот способ имеет ряд преимуществ перед остальными:

* наглядное отображение базовых конструкций алгоритма
* концентрация внимания на структуре алгоритма
* использование принципа блочности при коллективном решении сложной задачи
* преобразование алгоритма методом укрупнения (сведения к единому блоку) или детализации (разбиения на ряд блоков)
* быстрая проверка разработанного алгоритма

*В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т. п.) соответствует геометрическая фигура, называемая****блочным символом.***

Блочные символы соединяются линиями переходов (стрелками), определяющими очередность выполнения действий.

*Условные графические изображения, используемые при построении схем, называются****символами.***

Блок-схема выстраивается в одном направлении в порядке выполнения действий:

1. сверху вниз
2. слева направо

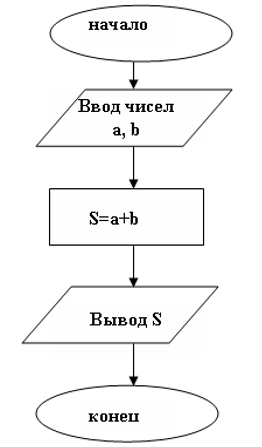
Для наглядности операции разного вида изображаются в схеме различными геометрическими фигурами – блоками (табл. 1).



1. ***Терминатор (пуск-остановка).*** Элемент отображает вход из внешней среды или выход из нее (наиболее частое применение − начало и конец алгоритма). Внутри фигуры записывается соответствующее действие - начало/конец.
2. ***Функциональный блок (операторный блок, процесс).*** Выполнение одной или нескольких операций, обработка данных любого вида (изменение значения данных, формы представления, расположения). Внутри фигуры записывают непосредственно сами операции. В этом блоке знак « = » означает не математическое равенство, а операцию присваивания.
3. ***Блок ввода/вывода.*** Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет типа носителя данных.
4. ***Альтернативный блок (уловный блок, условие, решение).***   
   Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определенных внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента. Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трех, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Соответствующие результаты вычислений могут записываться рядом с линиями, отображающими эти пути.
5. ***Блок цикла.*** Условия цикла и приращения записываются внутри символа цикла − в зависимости от типа организации цикла. Для изображения итерационного цикла на блок-схеме вместо данного символа используют символ решения, указывая в нем условие, а одну из линий выхода замыкают выше в блок-схеме (перед операциями цикла).
6. ***Блок предопределенного процесса (подпрограммы).***Символ отображает выполнение процесса, состоящего из одной или нескольких операций, который определен в другом месте программы (в подпрограмме, модуле, вспомогательном алгоритме). Внутри символа записывается название процесса и передаваемые в него данные.
7. ***Соединительный блок (разрыв).*** Используется для обрыва линии и продолжения ее в другом месте (пример: разделение блок-схемы, занимающей более одного листа). Соответствующие соединительные символы должны иметь одно (при том уникальное) обозначение.
8. ***Блок комментарий.*** Позволяет включать в схемы алгоритмов пояснения к функциональным блокам. Частое использование комментариев нежелательно, т. к. это усложняет (загромождает) схему, делает ее менее наглядной

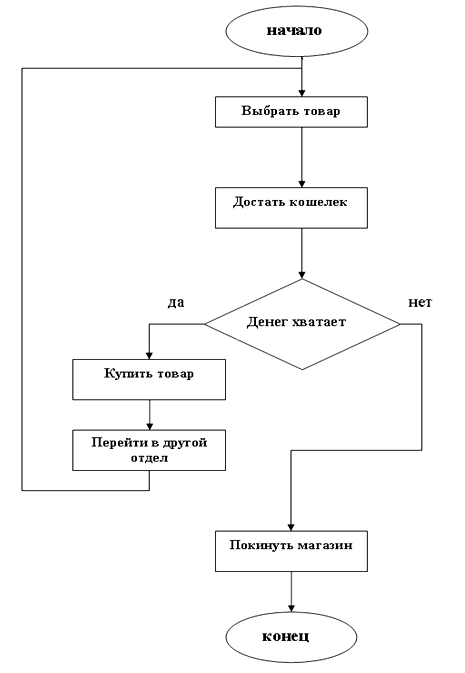
***Пример 9.****Составить алгоритм сложения двух чисел*

Блок-схема:



***Пример 10.****Составить алгоритм совершения покупок в магазине.*

Блок-схема:

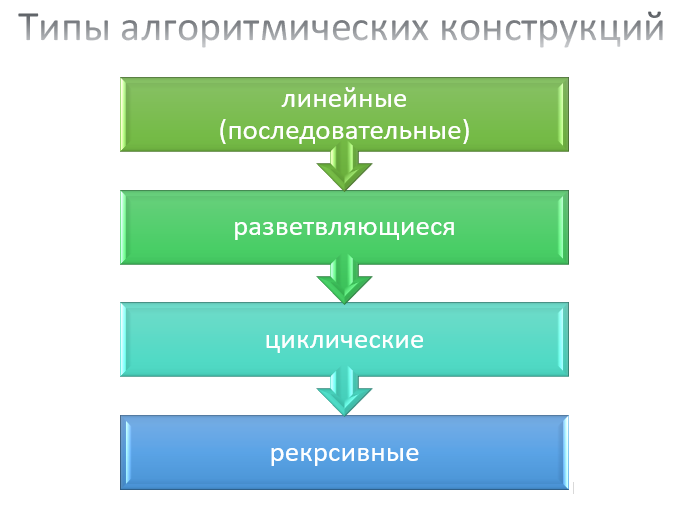


**2.  Типы алгоритмических конструкций**

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных базовых (т. е. основных) элементов. Эти элементарные шаги объединяются в алгоритмические конструкции.

Для решения любой задачи могут быть построены несколько алгоритмов, приводящих к получению результата ее решения. Из всех возможных алгоритмов следует выбирать наилучший по разным критериям: по точности решения задачи, временным затратам, количеству этапов в алгоритме, их простоте и т. д.

В зависимости от особенностей своего построения алгоритмы можно разделить четыре группы.



**1. Линейная алгоритмическая конструкция**(линейный алгоритм)

***Линейный (последовательный)****алгоритм выполняется в естественном порядке его написания и не содержит разветвлений и повторений; все этапы решения задачи выполняются ровно один раз и строго последовательно.*

**Примерами линейных алгоритмов являются:**

* алгоритм отпирания дверей - достать ключ, вставить ключ в замочную скважину, открыть замок;
* алгоритм заваривания чая - достать чайник, насыпать в него чай, залить кипятком, настоять 5-10 мин.

Алгоритм линейной структуры применяется при вычислении арифметического выражения, если в нем используются только простейшие алгебраические действия. Структура линейного алгоритма представлена на рисунке 1.

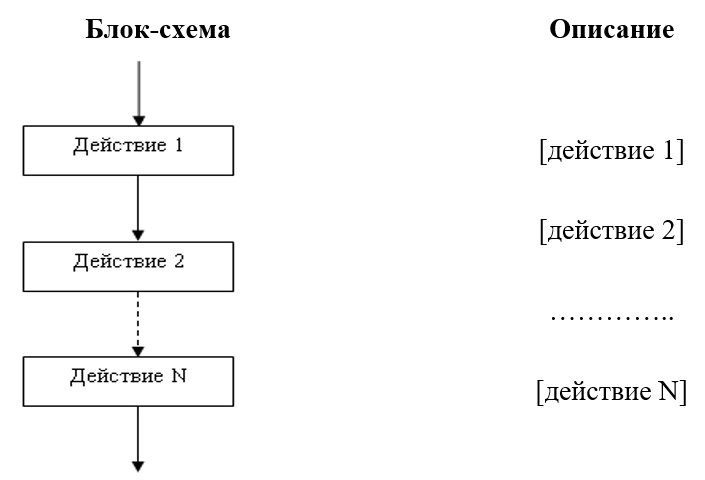


Рис. 1

Прямоугольник, изображенный на рисунке, может представлять как одну команду, так и множество операторов, необходимых для выполнения обработки данных.

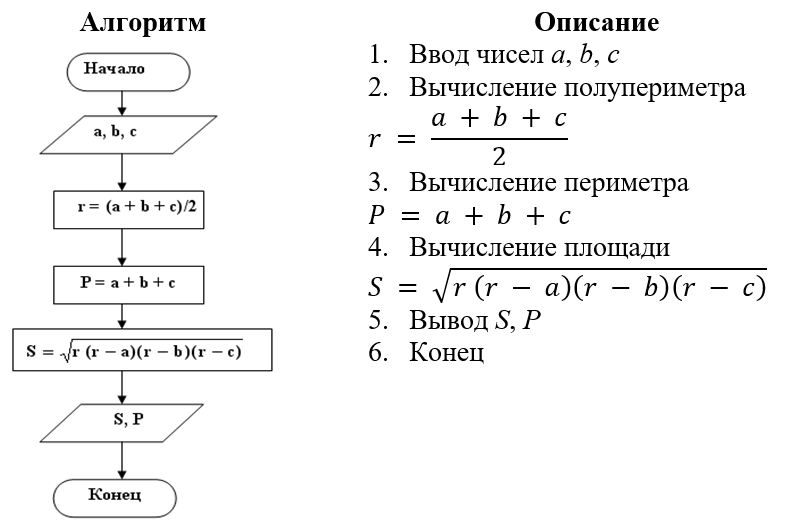
***При разработке линейного алгоритма необходимо учитывать:***

* *что данный вид алгоритма является простейшим*
* *используется для реализации простых вычислений по формулам*
* *инструкции в нем выполняются последовательно, одна за другой*

***Пример 1.****Составить алгоритм вычисления площади и периметра треугольника, если известны длины трех его сторон*

Входные данные:***a*, *b*, *c*** (длины сторон треугольника)

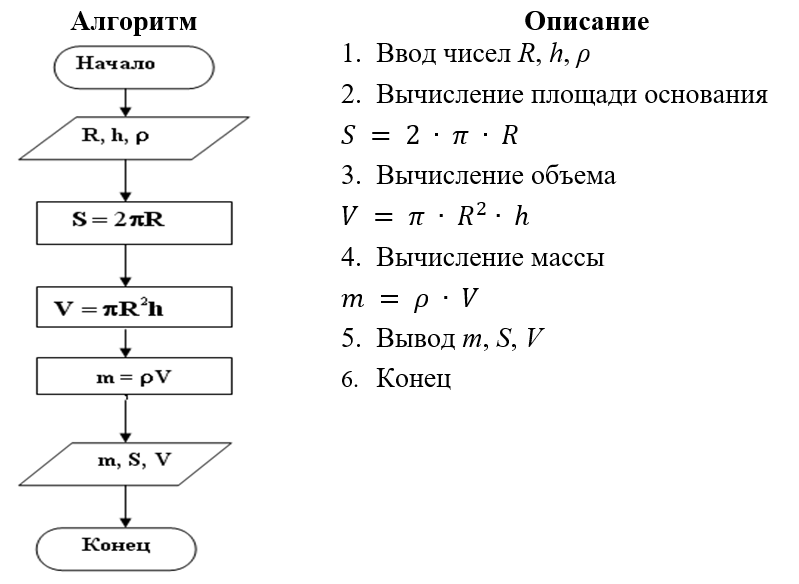
Выходные данные:***S*, *P*** (площадь и периметр треугольника)



***Пример 2.****Составить алгоритм вычисления объема, массы и площади основания цилиндрического тела, если известны его плотность и геометрические размеры: радиус основания и высота*

Входные данные: ***R***(радиус основания цилиндра), ***h***(высота цилиндра), ***ρ*** (плотность материала)

Выходные данные: ***m*** (масса), V (объем), ***S*** (площадь основания)



**2. Разветвляющаяся алгоритмическая конструкция**

На практике редко удается представить схему алгоритма решения задачи в виде линейной структуры. Задача может содержать условие, в зависимости от которого вычислительный процесс идет по той или иной ветви, т. е. появляется **ветвление** алгоритма.

***Разветвляющейся (ветвящейся)****называется конструкция, которая обеспечивает выбор между двумя альтернативами в зависимости от значения входных данных.*

*Или*

***Разветвляющийся алгоритм****— это алгоритм, в котором в зависимости от условия выполняется либо одна, либо другая последовательность действий.*

Примеры разветвляющих алгоритмов:

* **если**пошел дождь, **то**надо открыть зонт;
* **если**билет в театр стоит не больше ста рублей, **то**купить билет и занять свое место в зале, **иначе**(если стоимость билета больше 100 руб.) вернуться домой.

В общем случае количество ветвей в таком алгоритме не обязательно равно двум. При каждом конкретном наборе входных данных разветвляющийся алгоритм сводится к линейному.

**Структура «ветвление» существует в двух основных вариантах.**

***1. Полное ветвление*** – предполагает выполнение действий для обеих веток в алгоритме (рис. 1):

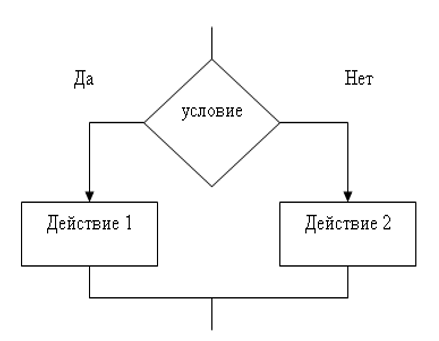


Рис. 1

**если** [условие],**то** [действие 1], **иначе** [действие 2]

***2. Неполное ветвление* –**предполагает выполнение действий только на одной ветви алгоритма, вторая отсутствует (рис. 2):

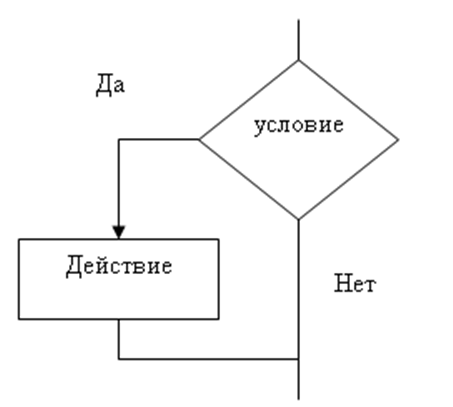


Рис. 2

**если** [условие],**то** [действие]

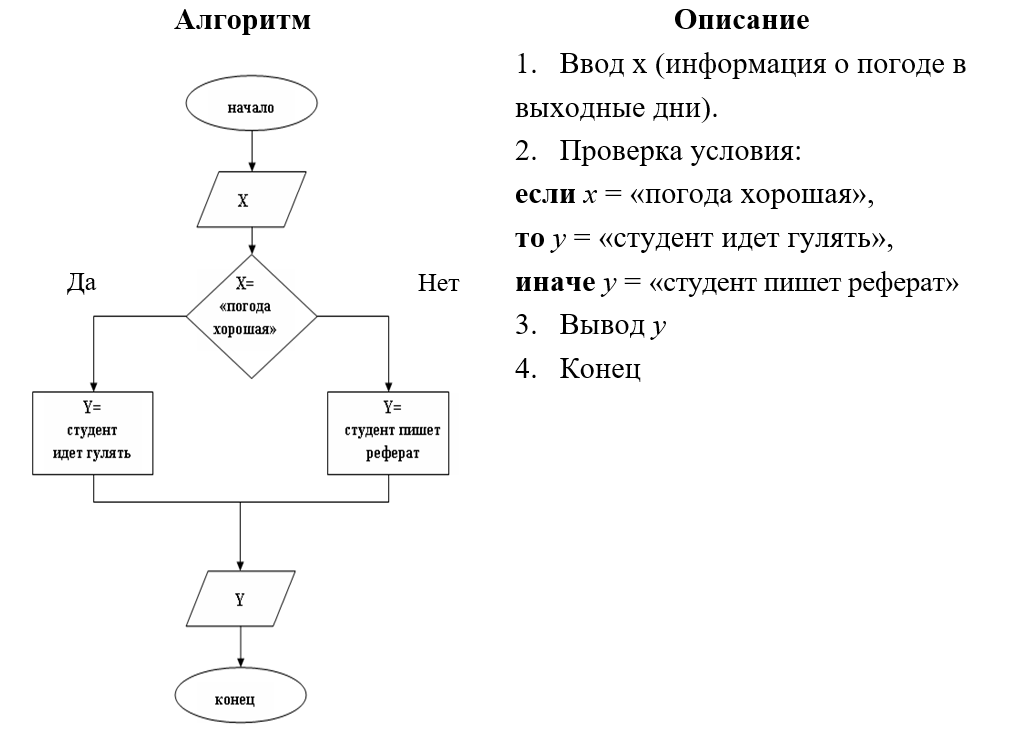
***При разработке разветвляющегося алгоритма необходимо учитывать:***

* *что данный вид алгоритма применяется при наличии операций условного перехода;*
* *чаще используется для вычислений функций, заданных несколькими арифметическими выражениями (формулами);*
* *инструкции в нем выполняются в зависимости от значения условия.*

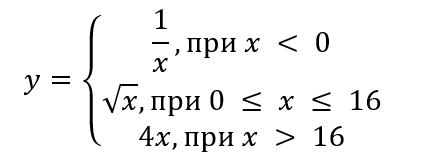
***Пример 1.****Составить алгоритм планирования выходного дня студентом: если будет хорошая погода, он пойдет гулять, а если плохая - будет писать реферат*

Входные данные:***x*** (информация о погоде)

Выходные данные:***y***(результат прошедшего выходного дня)



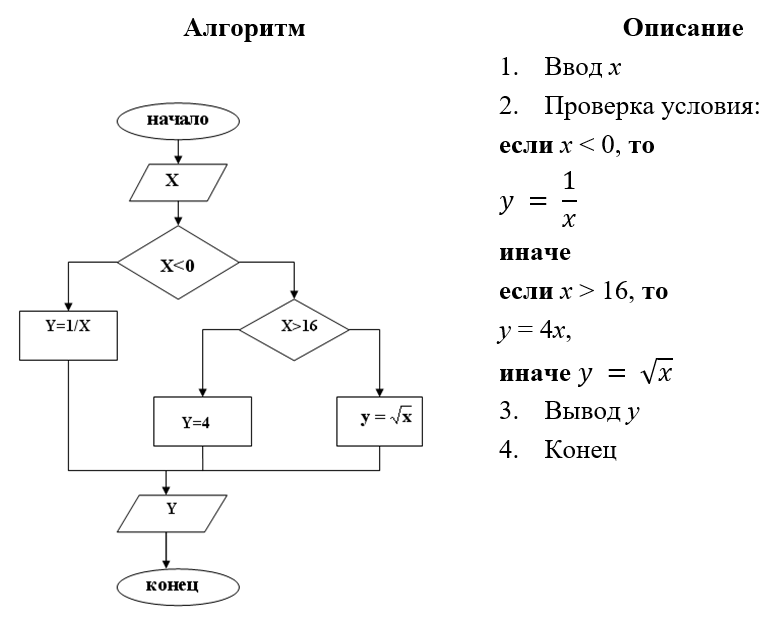
***Пример 2.****Составить алгоритм вычисления значения функции:*



Нахождение значения функции усложняется тем, что на области определения функция состоит из элементарных, непрерывных на заданном участке, функций. Прежде чем вычислить значение *y*, необходимо проверить принадлежность введенного **х** одному из заданных участков.

Входные данные:***x***

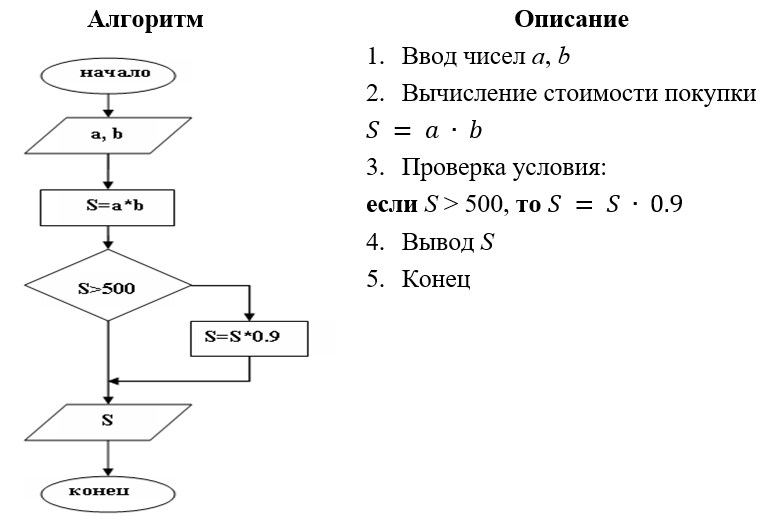
Выходные данные:***y***



***Пример 3.****Составить алгоритм вычисления стоимости покупки с учетом скидки: при покупке товара на сумму больше 500 руб. предоставляется скидка 10 %.*

Входные данные:***a*** (цена единицы товара), ***b*** (количество единиц товара)

Выходные данные: **s** (сумма покупки)



**3. Циклическая алгоритмическая конструкция**

При решении задач часто приходится проверять одно условие для нескольких значений или повторять некоторые действия несколько раз. Алгоритмы, отдельные действия которых многократно повторяются, называются ***алгоритмами циклической структуры*.**

***Циклической****называется конструкция, в которой некая, идущая подряд группа действий (шагов), может выполняться несколько раз, в зависимости от входных данных или условия задачи*

 Она позволяет существенно сократить объем алгоритма, представить его компактно за счет организации повторений большого числа одинаковых вычислений над разными данными для получения необходимого результата.

*Совокупность повторяющихся действий алгоритма называют****циклом***

*Группа повторяющихся действий на каждом шагу цикла называется****телом цикла***

**Примеры циклических алгоритмов:**

* покраска забора (повторяющаяся покраска каждого элемента)
* прием документов у абитуриентов в приемной комиссии
* опрос студентов
* чередование дней недели или времен года.

***Циклический алгоритм включает в себя:***

* *подготовку цикла – действия, связанные с заданием исходных данных, используемых в цикле;*
* *тело цикла – повторяющиеся действия для вычисления искомых величин, а также подготовка значений, необходимых для повторного выполнения действий в теле цикла;*
* *условия продолжения цикла – действия, определяющие необходимость дальнейшего выполнения тела цикла.*

Существует несколько видов циклических конструкций, с помощью которых можно организовать циклы (рис. 1).

**Классификация циклических конструкций**

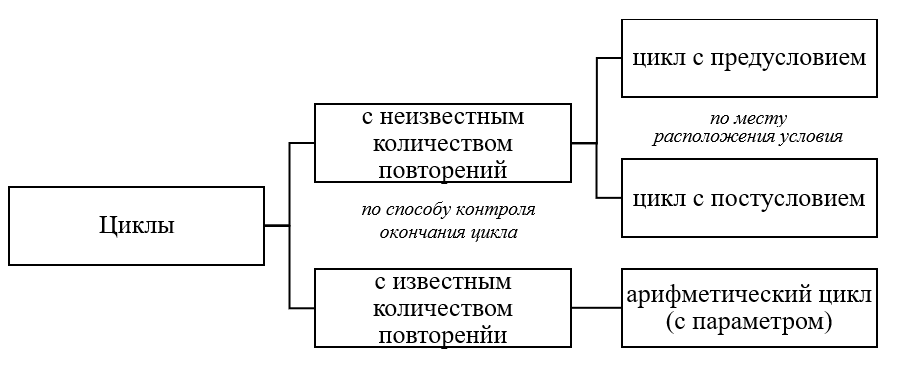


Рис. 1

***При разработке циклического алгоритма необходимо учитывать:***

* *использование циклов позволяет существенно сократить схему алгоритма*
* *при организации цикла следует особое внимание уделить правильному оформлению изменения параметра цикла, потому что ошибка на этом этапе может привести к «зацикливанию» вычислений*
* *количество повторений в цикле зависит от входных данных или условий задачи*
* *для завершения в теле цикла должны быть инструкции, выполнение которых влияет на завершение цикла*
* *он чаще используется для табулирования функций, для решения задач с повторными вычислениями*



Циклы, для которых число повторений определяется в ходе выполнения цикла, называются ***итерационными*** (циклы с неизвестным числом повторений). В этом случае одно повторение цикла называется ***итерацией*.**

**4. Рекурсивная алгоритмическая конструкция**(нелинейные, подчиненные алгоритмы)

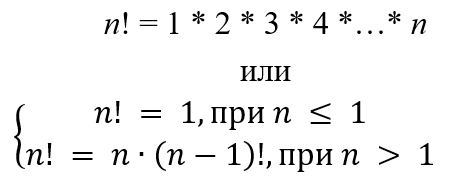
Как правило, в основе такого алгоритма лежит рекурсивное определение какого-то понятия.

***Рекурсивным****называется алгоритм, который в процессе выполнения на каком-либо шаге прямо или косвенно обращается сам к себе*

 Отличие рекурсивного алгоритма от циклического заключается в том, что в определенной точке алгоритма реализуется тот же самый алгоритм. Рекурсивный объект частично определяется через себя. Преимущество рекурсивного определения объекта заключается в том, что такое конечное определение теоретически способно описывать бесконечно большое число объектов. В рекурсивном определении должно присутствовать ограничение, ***граничное условие***, при выходе на которое рекурсивный возврат заканчивается.

**Пример рекурсивного определения:**

* определение факториала числа ***n!***

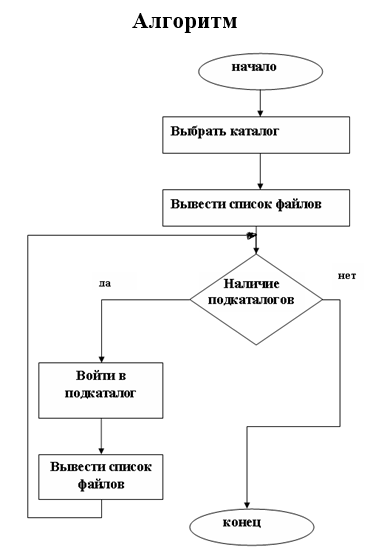


  Граничным условием является условие https://ucarecdn.com/c6b6133c-541c-4436-a531-64cc9421c77c/

***Пример 1.****Составить алгоритм поиска всех файлов с именем «к\*т» в каталоге и во всех подкаталогах этого каталога*

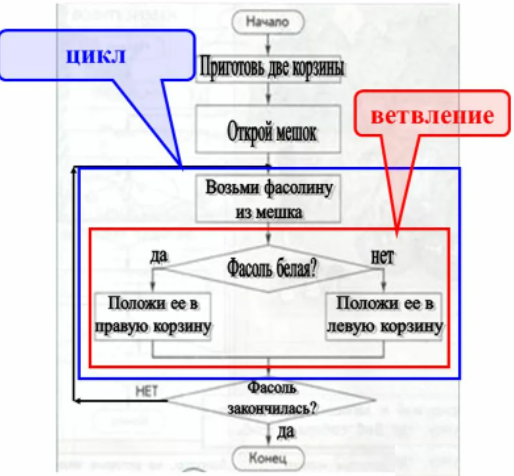
**Описание**

1. Открыть каталог
2. Вывести список всех файлов, удовлетворяющих критерию запроса: имя = <к\*т>
3. Если в каталоге есть подкаталоги, то выполнить п. 1, иначе конец.



**3. Комбинированные алгоритмы**

**Комбинированные алгоритмы**



На практике часто встречаются задачи, алгоритм решения которых распадается на части, фрагменты и каждый фрагмент представляет собой алгоритм одного из трех описанных в теме "Основные алгоритмические конструкции" типов.

*Алгоритм, который содержит несколько структур одновременно, называется****комбинированным.***

**В теории алгоритмов доказано:**

***Любой, сколь угодно сложный алгоритм может быть составлен из трех основных алгоритмических структур:***

* *линейной*
* *ветвления*
* *цикла*

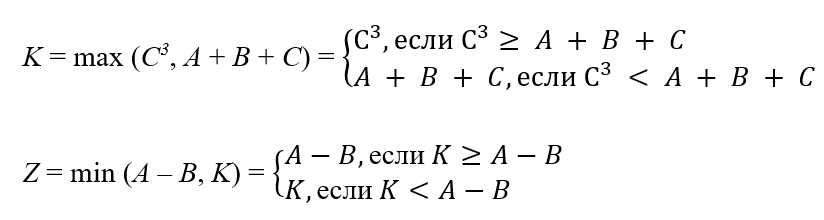
 Алгоритмы решения сложных задач могут включать все перечисленные структуры либо комбинации некоторых из них.

***Пример 1.****Составить алгоритм вычисления значения переменной****Z****, где****Z****= min (****A****–****B****, max (****C3****,****A****+****B****+****С****)).****A, B, C****– действительные числа.*

Возможны три варианта ответа:

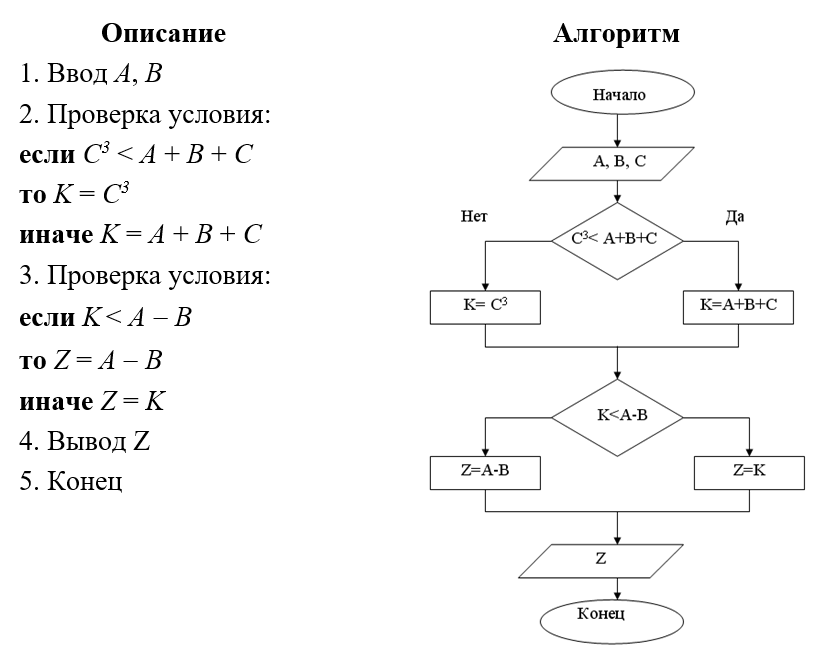
1. **(*A* – *B*)**
2. ***C3***
3. **(*A* + *B* + *С*)**

Обозначим:



Входные данные:***А*, *B*, *C***

Выходные данные:***K*, *Z***



**Вложенные циклы**

Частным случаем комбинированных алгоритмов являются алгоритмы, в которых необходимо использовать несколько циклов, изменяющихся одновременно. Например, внутри одного цикла могут находиться один или несколько других циклов.

*Такие комбинации называются****"цикл в цикле"****или****вложенные циклы:***

* *охватывающий цикл называется****внешним***
* *вложенные в него циклы –****внутренними (вложенными)***

Область действия внутреннего цикла должна полностью находиться в области внешнего цикла, т. е. циклы не должны пересекаться (рис. 1).



Правила организации как внешнего, так и внутреннего циклов такие же, как и правила организации простого цикла.

Параметры внешнего и внутреннего должны иметь разное имя и изменяться не одновременно, т. е. при одном значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения. Перед циклом необходимо задавать начальные значения параметров, а внутри - вычислять текущие.

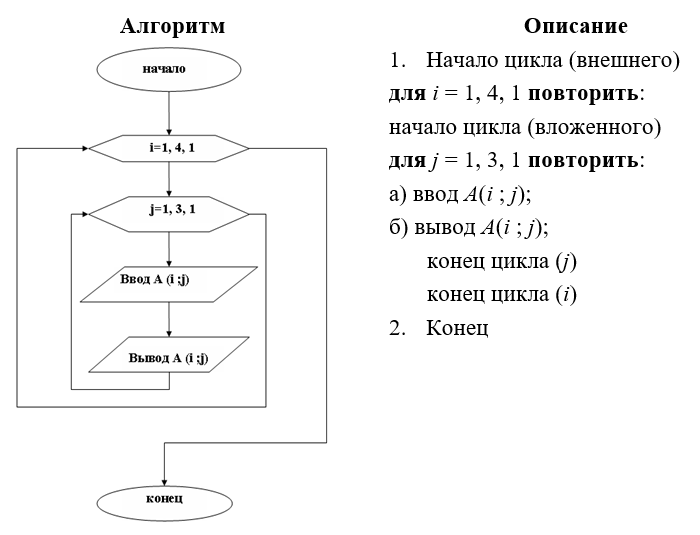
***Пример 1.****Составить алгоритм заполнения таблицы данными о зарплате сотрудников предприятия за первый квартал 2022 года.*

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО сотрудника | Январь | Февраль | Март |
| Иванов | 1000 | 1200 | 1300 |
| Петров | 1100 | 1000 | 1200 |
| Сидоров | 2000 | 2100 | 2200 |
| Васечкин | 3000 | 3100 | 3300 |

Обозначим вводимые данные как элементы двумерного массива  
{*A* (*i*; *j*)}, где *i* = 1, 2, 3, 4 (номер строки)  
*j*= 1, 2, 3     (номер столбца)

|  |  |  |
| --- | --- | --- |
| A(1;1)=1000 | A(1;2)=1200 | A(1;3)=1300 |
| A(2;1)=1100 | A(2;2)=1000 | A(2;3)=1200 |
| A(3;1)=2000 | A(3;2)=2100 | A(3;3)=2200 |
| A(4;1)=3000 | A(4;2)=3100 | A(4;3)=3300 |

Задача сводится к заполнению двумерного массива.  
Входные данные:***i, j, A*(*i*; *j*)**  
Выходные данные: ***A*(*i*; *j*)**



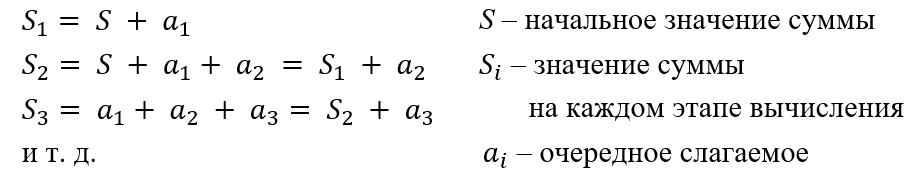
**Организация счётчиков**

Для построения алгоритма решения таких задач используется цикл с параметром, т. к. число шагов цикла известно. В алгоритмах вычисления сумм, произведений, количеств, пределов, последовательностей особенностью является содержание тела цикла.

**1. Вычисление суммы**

Традиционный алгоритм для решения задачи **с*уммирования***состоит в последовательном суммировании элементов числового набора .

При вычислении суммы используется прием накопления:



Вычисление суммы сводится к ее накоплению в виде значения переменной в цикле, в котором вычисляются соответствующие слагаемые. При этом вновь вычисленное слагаемое прибавляется к сумме предыдущих слагаемых, т. е. в цикле последовательно вычисляются все промежуточные суммы.

После первого выполнения цикла первая промежуточная сумма должна быть равна значению первого слагаемого. Следовательно, начальное значение***S*** должно быть равно нулю.

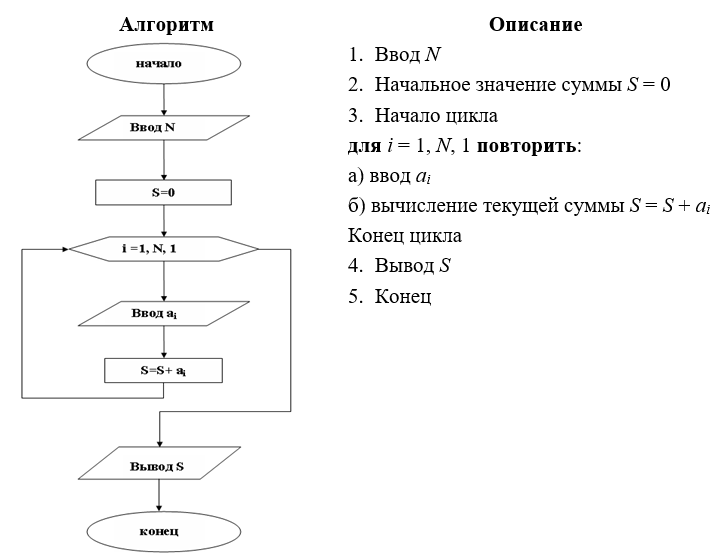
***Правило суммирования:***

* *начальное значение суммы****S = 0***
* *в теле циклической конструкции применяется****формула для накопления суммы S = S + a,*** *где****a****– очередное слагаемое,****S****– промежуточная сумма*

***Пример 1.****Составить алгоритм вычисления общей суммы выплаченной премии всем сотрудникам фирмы*

Входные данные:***N*** (количество сотрудников фирмы)  
                                  ***аi***(размер премии каждого сотрудника)  
                                  ***i***(номер сотрудника, параметр).

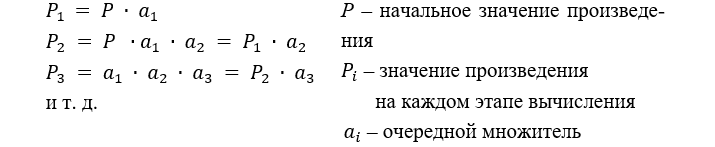
Выходные данные: ***S*** (общая сумма выплат)



**2. Вычисление произведения**

Традиционный алгоритм для решения задачи **вычисления произведения** состоит в последовательном умножении элементов числового набора .

При вычислении произведения используется также прием накопления:



Вычисление произведения сводится к его накоплению в цикле в виде значения переменной, при этом в цикле вычисляются последовательно все промежуточные произведения.

После первого выполнения цикла первая промежуточная величина должна быть равна значению первого множителя. Следовательно, начальное значение ***P***, которое задается перед циклом, должно быть равно единице.

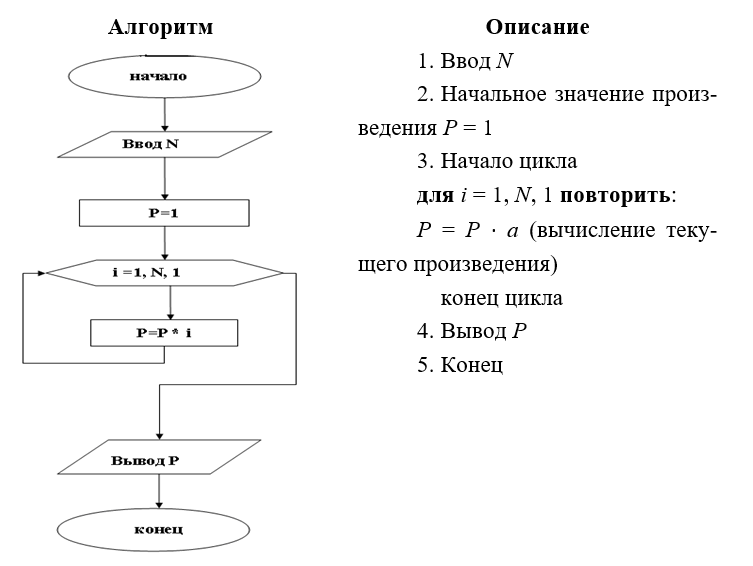
***Правило умножения:***

* *начальное значение произведения****Р = 1***
* *в теле циклической конструкции применяется****формула для накопления произведения P = P × a****,  
  где****a****– очередной множитель,****P****– промежуточное произведение*

***Пример 2.****Составить алгоритм вычисления N! (N! = 1 × 2 × 3 × 4 ×…×N)*

Входные данные:***N***(натуральное число); ***i***(параметр)

Выходные данные:***P* *= N*!**



**3. Подсчет количества элементов**

При вычислении количества элементов используется прием накопления, как и при вычислении суммы и произведения. Подсчет количества элементов сводится к вычислению количества натуральных чисел в последовательности 1, 2, 3, 4 и т. д.

***Счетчик***искомых элементов обозначается ***k*.**Процесс является циклическим, так как каждый раз совершается одно и тоже действие: увеличение предыдущего натурального числа на 1.

При первом подсчете должно получиться значение *k*, равное единице. Следовательно, начальное значение счетчика должно быть равно нулю.

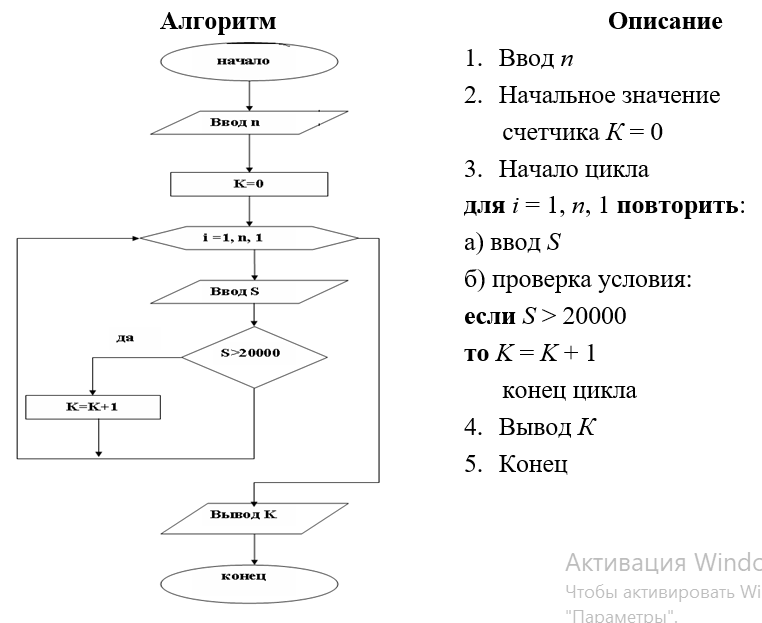
***Правило счетчика:***

* *начальное значение счетчика k = 0*
* *в теле циклической конструкции применяется формула для накопления счетчика k = k + 1, где k – промежуточное значение счетчика*

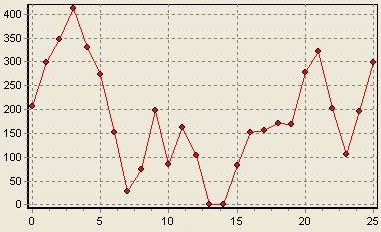
***Пример 3.****Составить алгоритм подсчета количества сотрудников фирмы, зарплата которых превышает 20 тыс. руб. (составить алгоритм подсчета количества чисел больших 20 среди n заданных)*

Входные данные: ***n***(число сотрудников); ***S*** (размер зарплаты);***i***(номер очередного сотрудника, параметр)

Выходные данные: счетчик***К*** (количество сотрудников, удовлетворяющих условию)



**Табулирование** **функции**



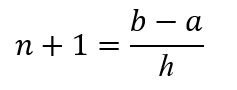
***Табулирование****- вычисление значений функции при изменении аргумента от некоторого начального значения до некоторого конечного значения с определенным шагом*

 При этом вид функции и шаг может меняться. Табулирование применяется для составления всевозможных таблиц, которыми могут быть как абстрактная таблица значений математической функции, так и конкретная таблица стоимости товара или платежей, совершенных абонентом сотового оператора.

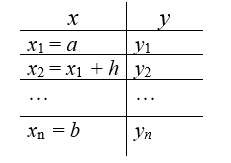
**Общая математическая формулировка такой задачи:**

***Вычислить и напечатать таблицу значений аргумента x и функции y = f(x) при изменении аргумента x на отрезке [a; b] с шагом h, где:***

* *a – начало отрезка*
* *b – конец отрезка*
* *h – шаг (изменения аргумента)*
* *n – количество шагов*
* *(n + 1) – количество точек*

 Параметры связаны следующей формулой: 

Результатом должна быть следующая таблица пар значений (*x*; *y*):



**В общем виде алгоритм можно описать так:**

* *определяется переменная (обычно x)*
* *перед циклом задается начальное значение переменной, условием окончания цикла является достижение переменной конечного значения*
* *в теле цикла на каждом шаге вычисляется значение функции y, зависящее от переменной x (формируется строка таблицы)*
* *в конце каждого шага значение переменной изменяется на h, где h - заданный шаг изменения, т. е. x = x + h*

Алгоритм табулирования содержит все основные конструкции: линейную, ветвление, цикл.

**1. Табулирование функции, заданной на интервале одной формулой *y = f*(*x*)**

 Функция, как правило, определена на заданном участке. Требуется ***n*** раз произвести вычисления одного выражения при разных значения *х* (рис. 1).

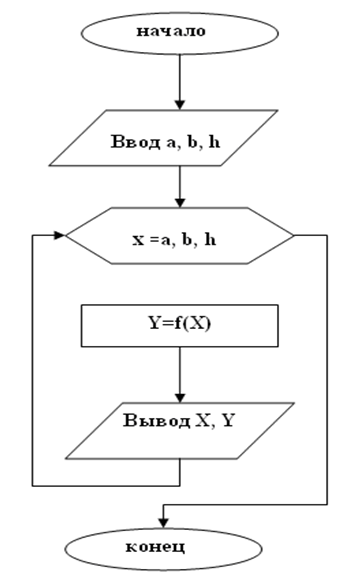


Рис .1

***Алгоритм решения такой задачи включает следующие этапы:***

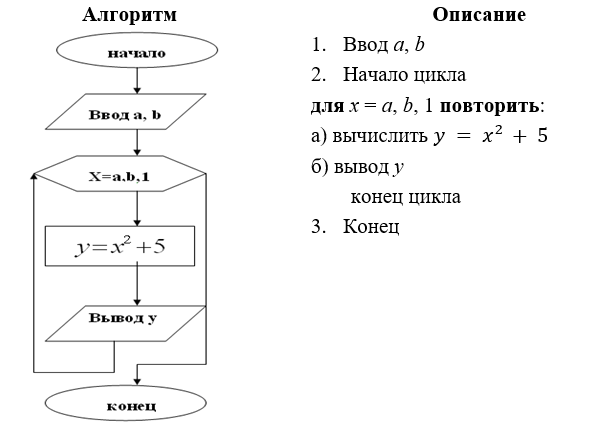
* *ввод исходных данных a, b, h;*
* *организация цикла по x от начального значения a до конечного значения b с шагом h;*
* *вычисление y = f(x);*
* *печать x, y;*
* *конец цикла.*

В зависимости от условий конкретной задачи к перечисленным этапам могут добавляться другие вычисления.

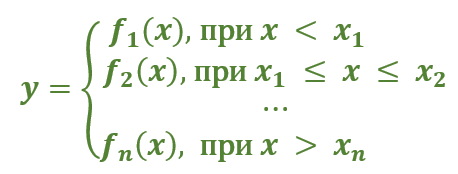
***Пример 1.****На отрезке [a ; b] составить алгоритм табулирования функции . Значения a, b могут изменяться*

Входные данные:***х*** (переменная, параметр цикла); ***а*** (начальное значение параметра); ***b*** (конечное значение параметра)

Выходные данные:**y**



**2. Табулирование функции, заданной на интервале несколькими формулами**



  Функция разделена на промежутки, на которых она задана элементарными функциями. Каждая из этих функций, как правило, определена на заданном интервале. Такая задача отличается тем, что необходимо, изменяя значение аргумента, проверять, в какой из определенных промежутков он попадает, в качестве значения функции *y* выбирать формулу, соответствующую промежутку (рис. 2).

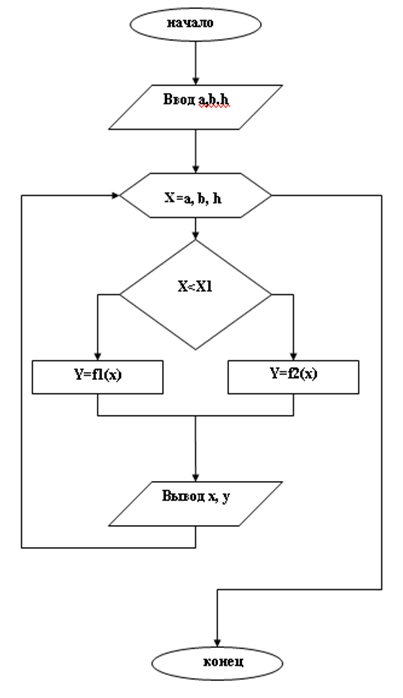


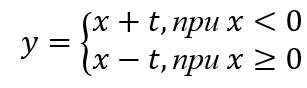
Рис. 2

***Алгоритм решения такой задачи состоит из следующих этапов:***

* *ввод исходных данных****a, b, h***
* *организация цикла по x от начального значения a до конечного значения****b****с шагом****h***
* *проверка условия: определяется интервал, в котором находится****x***
* *вычисление****y1 = f1(x)***
* *при необходимости пп. 3–4 повторяются*
* *печать****x, y***
* *конец цикла*

В зависимости от условий конкретной задачи, к перечисленным этапам могут добавляться другие вычисления.

***Пример 2.****Составить алгоритм табулирования функции на отрезке [a ; b], если а = -2, h = 0,2, n = 11*



Входные данные:***х***(переменная, параметр цикла); ***а*** (начальное значение параметра); ***n*;*h***

Выходные данные:***y***

