**Основы алгоритмизации и технологии программирования**

**Понятие алгоритма и его свойства**

        Каждый из нас постоянно решает множество задач: как быстрее обраться на работу, как лучше спланировать дела текущего дня и многие другие. Некоторые задачи мы решаем автоматически, так как на протяжении многих лет привыкли к их выполнению, другие требуют длительного размышления над решением, но в любом случае, решение каждой задачи всегда делится на простые действия.

       Алгоритм – описанная на некотором языке точная конечная система правил, определяющая содержание и порядок действий над некоторыми объектами, строгое выполнение которых дает решение, поставленной задачи. Понятие алгоритма, являющееся фундаментальным в математике и информатике, возникло задолго до появления средств вычислительной техники. Слово «алгоритм» появилось в средние века, когда европейцы познакомились со способами выполнения арифметических действий в десятичной системе счисления, описанными узбекским математиком Муххамедом бен Аль-Хорезми («аль-Хорезми» - человек из города Хорезми); в настоящее время город Хива в Хорезмской области Узбекистана). Слово алгоритм – есть результат европейского произношения слов аль-Хорезми. Первоначально под алгоритмом понимали способ выполнения арифметических действий над десятичными числами. В дальнейшем это понятие стали использовать для обозначения любой последовательности действий, приводящей к решению поставленной задачи.

        Любой алгоритм существует не сам по себе, а предназначен для определенного исполнителя (человека, робота, компьютера, языка программирования и т.д.). Свойством, характеризующим любого исполнителя, является то, что он умеет выполнять некоторые команды. Совокупность команд, которые данный исполнитель умеет выполнять, называется системой команд исполнителя. Алгоритм описывается в командах исполнителя, который будет его реализовывать. Объекты, над которыми исполнитель может совершать действия, образуют так называемую среду исполнителя. Исходные данные и результаты любого алгоритма всегда принадлежат среде того исполнителя, для которого предназначен алгоритм.

        Значение слова «алгоритм» очень схоже со значениями слов «рецепт», «метод», «процесс». Однако, в отличие от рецепта или процесса, алгоритм характеризуется следующими свойствами: дискретностью, массовостью, определенностью, результативностью, формальностью.

        Дискретность (разрывность – противоположно непрерывности) – это свойство алгоритма, характеризующее его структуру: каждый алгоритм состоит из отдельных законченных действий, говорят: «Делится на шаги».

      Массовость – применимость алгоритма ко всем задачам рассматриваемого типа, при любых исходных данных. Например, алгоритм решения квадратного уравнения в области действительных чисел должен содержать все возможные исходы решения, т.е., рассмотрев значения дискриминанта, алгоритм находит либо два различных корня уравнения, либо два равных, либо делает вывод о том, что действительных корней нет.

    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. Помните сказку про Ивана-царевича? «Шел Иван-царевич по дороге, дошел до развилки. Видит большой камень, на нем надпись: «Прямо пойдешь – голову потеряешь, направо пойдешь – жену найдешь, налево пойдешь – разбогатеешь. Стоит Иван и думает, что дальше делать». Таких инструкций алгоритм содержать не может.

        Результативность – свойство, состоящее в том, что любой алгоритм должен завершаться за конечное (может быть очень большое) число шагов. Вопрос о рассмотрении бесконечных алгоритмов остается за рамками теории алгоритмов.

       Формальность – это свойство указывает 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. Рассуждать «что, как и почему» должен разработчик алгоритма, а исполнитель формально (не думая) поочередно исполняет предложенные команды и получает необходимый результат.

**Способы описания алгоритмов**

        Рассмотрим следующие способы описания алгоритма: словесное описание, псевдокод, блок-схема, программа.

        Словесное описание представляет структуру алгоритма на естественном языке. Например, любой прибор бытовой техники (утюг, электропила, дрель и т.п.) имеет инструкцию по эксплуатации, т.е. словесное описание алгоритма, в соответствии которому данный прибор должен использоваться.

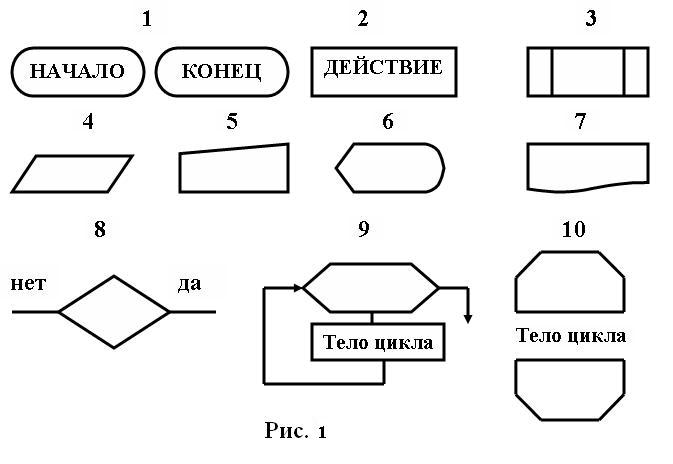
        Никаких правил составления словесного описания не существует. Запись алгоритма осуществляется в произвольной форме на естественном, например, русском языке. Этот способ описания не имеет широкого распространения, так как строго не формализуем (под «формальным» понимается то, что описание абсолютно полное и учитывает все возможные ситуации, которые могут возникнуть в ходе решения); допускает неоднозначность толкования при описании некоторых действий; страдает многословностью.

       Псевдокод – 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.

        Строгих синтаксических правил для записи псевдокода не существует. Это облегчает запись алгоритма при проектировании и позволяет описать алгоритм, используя любой набор команд. Однако в псевдокоде обычно используются некоторые конструкции, присущие формальным языкам, что облегчает переход от псевдокода к записи алгоритма на языке программирования. Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором используемых слов и конструкций.

       Блок-схема – описание структуры алгоритма с помощью геометрических фигур с линиями-связями, показывающими порядок выполнения отдельных инструкций. Этот способ имеет ряд преимуществ. Благодаря наглядности, он обеспечивает «читаемость» алгоритма и явно отображает порядок: выполнения отдельных команд. В блок-схеме каждой формальной конструкции соответствует определенная геометрическая фигура или связанная линиями совокупность фигур.

         Рассмотрим некоторые основные конструкции, использующиеся для построения блок-схем (рис. 1).



(1) Блок, характеризующий начало/конец алгоритма (для подпрограмм – вызов/возврат);

(2) Блок - *процесс,*предназначенный для описания отдельных действий;

(3) Блок - *предопределенный процесс,*предназначенный для обращения к вспомогательным алгоритмам (подпрограммам);

(4) Блок - *ввода/вывода*с *неопределенного носителя;*

(5) Блок - ввод с клавиатуры;

(6) Блок - вывод на монитор;

(7) Блок - вывод на печатающее устройство;

(8) Блок – решение (проверка условия или условный блок);

(9) Блок, описывающий блок с параметром;

(10) Блок – границы цикла, описывающий циклические процессы типа: «цикл с предусловием», «цикл с постусловием»;

           Описания алгоритма в словесной форме, на псевдокоде или в виде блок-схемы допускают некоторый произвол при изображении команд. Вместе с тем она настолько достаточна, что позволяет человеку понять суть дела и исполнить алгоритм. На практике исполнителями алгоритмов выступают компьютеры. Поэтому алгоритм, предназначенный для исполнения на компьютере, должен быть записан на «понятном» ему языке, такой формализованный язык называют *языком программирования.*

*Программа –*описание структуры алгоритма на языке алгоритмического программирования. Программа на языке декларативного программирования представляет собой совокупность описанных знаний и не содержит явного алгоритма исполнения.

**Основные алгоритмические конструкции**

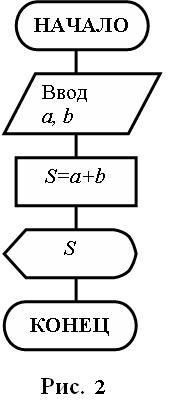
         Элементарные шаги алгоритма можно объединить в следующие алгоритмические конструкции: *линейные (последовательные), разветвляющиеся, циклические и рекурсивные.*

**Линейная алгоритмическая конструкция**

*Линейной*называют алгоритмическую конструкцию, реализованную в виде последовательности действий (шагов), в которой каждое действие (шаг) алгоритма выполняется ровно один раз, причем после каждого *i-* гo действия (шага) выполняется *(i+* 1)-е действие (шаг), если *i*-e действие – не конец алгоритма.

         Пример 1.

       Опишем алгоритм сложения двух чисел на псевдокоде в виде блок-схемы (рис. 2).



         Псевдокод:

Ввод двух чисел *а, b .*

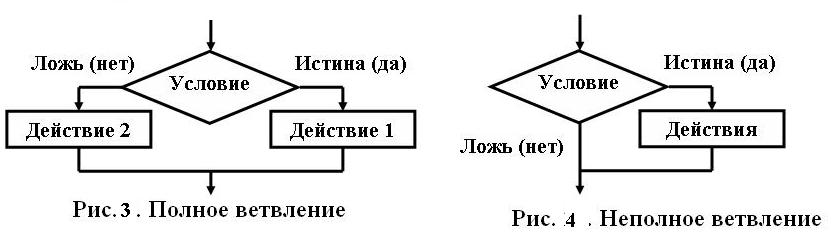
Вычисляем сумму S = *а*+ *b .*

Вывод S.

Конец.

**Разветвляющаяся алгоритмическая конструкция**

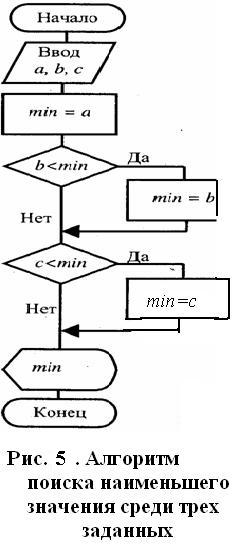
*Разветвляющейся*(или *ветвящейся)*называется алгоритмическая конструкция, обеспечивающая выбор между двумя альтернативами в зависимости от значения входных данных. При каждом конкретном наборе входных данных разветвляющийся алгоритм сводится к линейному. Различают неполное *(если – то)*и полное *(если – то – иначе)*ветвления. Полное ветвление позволяет организовать две ветви в алгоритме *(то*или *иначе),*каждая из которых ведет к общей точке их слияния, так что выполнение алгоритма продолжается независимо от того, какой путь был выбран (рис. 3). Неполное ветвление предполагает наличие некоторых действий алгоритма только на одной ветви *(то),*вторая ветвь отсутствует, т.е. для одного из результатов проверки никаких действий выполнять не надо, управление сразу переходит к точке слияния (рис. 4).



         Рассмотрим стандартный *алгоритм поиска наибольшего (наименьшего)*значения среди нескольких заданных. Основная идея алгоритма сводится к следующему: за наибольшее (наименьшее) принимаем значение любого из данных. Поочередно сравниваем оставшиеся данные с наибольшим (наименьшим). если окажется, что очередное значение входного данного больше (меньше) наибольшего (наименьшего), то наибольшему (наименьшему) присваиваем это значение. Таким образом, *сравнив* все входные данные, найдем наибольшее (наименьшее) среди них. Алгоритм использует неполное ветвление.

             Пример 2.

           Заданы три числа. Найти значение наименьшего из них Заданные числа обозначим: *а, b, с*; результирующее наименьшее – *min.*На рис. 5 представлена блок-схема алгоритма решения данной задачи.



**Алгоритмическая конструкция «Цикл»**

*Циклической*(или *циклом)*называют алгоритмическую конструкцию, в кoтoрoй некая, идущая подряд группа действий (шагов) алгоритма может выполняться несколько раз, в зависимости от входных данных или условия задачи. Группа повторяющихся действий на каждом шагу цикла называется *телом цикла.*Любая циклическая конструкция содержит себе элементы ветвящейся алгоритмической конструкции.

        Рассмотрим три типа циклических алгоритмов: *ц uкл*с *параметром*(который называют *арифметическим циклом*), *цикл с* *предусловием*и *цикл*с *постусловием*(их называют *итерационными)* .

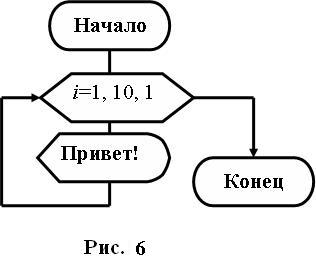
**Арифметический цикл**

        В арифметическом цикле число его шагов (повторений) однозначно определяется правилом изменения параметра, которое задается с помощью *начального (N)*и *конечного (К) значений параметра*и *шагом (h)*его изменения. Т.е., на первом шаге цикла значение параметра равно N, на втором – N + h, на третьем – N + 2h и т.д. На последнем шаге цикла значение параметра не больше *К*, но такое, что дальнейшее его изменение приведет к значению, большему, чем *К.*

        Пример 3.

      Вывести 10 раз слово «Привет!».

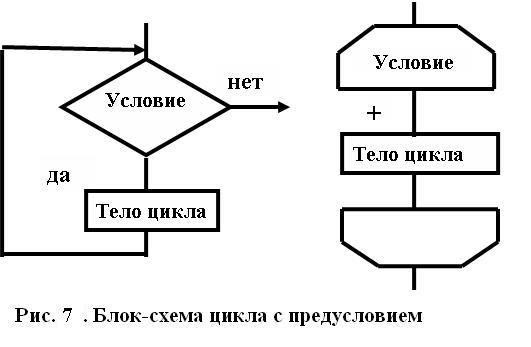
       Параметр цикла обозначим *i,*он будет отвечать за количество выведенных слов. При *i*=1 будет выведено первое слово, при *i*=2 будет выведено второе слова и т. д. Так как требуется вывести 10 слов, то последнее значение параметра *i*=10. В заданном примере требуется 10 раз повторить одно и то же действие: вывести слово «Привет!». Составим алгоритм, используя арифметический цикл, в котором правило изменения параметра i=1,10, 1. т. е. начальное значение параметра i=1; конечное значение i=10; шаг изменения h=1. На рис. 6 представлена блок-схема алгоритма решения данной задачи.



**Цикл с предусловием**

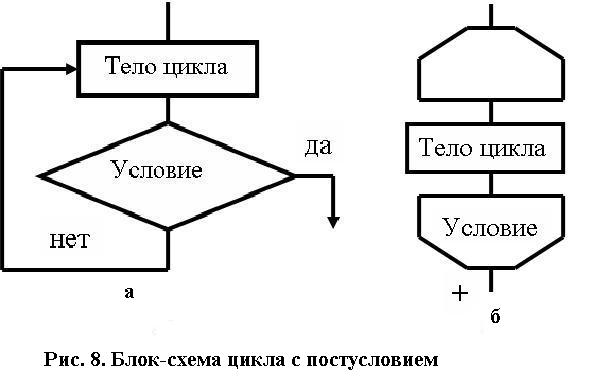
         Количество шагов цикла заранее не определено и зависит от входных данных задачи. В данной циклической структуре сначала проверяется значение условного выражения (условие) перед выполнением очередного шага цикла. Если значение условного выражения истинно, исполняется тело цикла. После чего управление вновь передается проверке условия и т.д. Эти действия повторяются до тех пор, пока условное выражение не примет значение ложь. При первом же несоблюдении условия цикл завершается.

         Блок-схема данной конструкции представлена на рис. 7 двумя способами: с помощью условного блока *а*и с помощью блока границы цикла *б.* Особенностью цикла с предусловием является то, что если изначально условное выражение ложно, то тело цикла не выполнится ни разу.



**Цикл с постусловием**

         Как и в цикле с предусловием, в циклической конструкции с постусловием заранее не определено число повторений тела цикла, оно зависит от входных данных задачи. В отличие от цикла с предусловием, тело цикла с постусловием всегда будет выполнено хотя бы один раз, после чего проверяется условие. В этой конструкции тело цикла будет выполняться до тех пор, пока значение условного выражения ложно. Как только оно становится истинным, выполнение команды прекращается. Блок-схема данной конструкции представлена на рис. 8 двумя способами: с помощью условного блока **а** и с помощью блока управления **б**.



**Рекурсивный алгоритм**

*Рекурсивным*называется алгоритм, организованный таким образом, что в процессе выполнения команд на каком-либо шаге он прямо или косвенно *обращается сам к себе.*

**Простые типы данных: переменные и константы**

         Реальные данные, которые обрабатывает программа, - это целые и вещественные числа, символы и логические величины. Эти простые типы данных называют *базовыми.*Все данные, обрабатываемые компьютером, хранятся в ячейках памяти компьютера, каждая из которых имеет свой адрес. Для того чтобы не следить за тем, по какому адресу будут записаны те или иные данные, в языках программирования используется понятие *переменной,*позволяющее отвлечься от адреса ячейки памяти и обращаться к ней с помощью имени *(идентификатора).*

*Переменная –*есть именованный объект (ячейка памяти), который может изменять свое значение. *Имя*переменной указывает на *зн ачение,*а способ ее хранения и адрес остаются скрытыми от программиста. Кроме имени и значения, переменная имеет *тип,*определяющий, какая информация находится в памяти. *Тип*переменной задает:

* используемый способ записи информации в ячейки памяти;
* необходимый объем памяти для ее хранения.

         Объем памяти для каждого типа определяется таким образом, чтобы в него можно было поместить любое значение из допустимого диапазона значений данного типа. Например, тип «байт» может принимать значения от О до 255, что в двоичном коде (255(10)=11111111(2)) соответствует ячейке памяти длиной в 8 бит (или 1 байт).

          В описанных выше алгоритмах (примеры 1-3) все данные хранятся в виде переменных. Например, инструкция «Ввод двух чисел *а, b*» означает введение пользователем значений двух переменных, а инструкция «К=К + 1» означает увеличение значения переменной К на единицу.

         Если переменные присутствуют в программе, на протяжении всего времени ее работы – их называют *статическими.*Переменные, создающиеся и уничтожающиеся на разных этапах выполнения программы, называют *динамическими.*

         Все остальные данные в программе, значения которых не изменяются на протяжении ее работы, называют *константами*или *постоянными.*Константы, как и переменные, имеют тип. Их можно указывать явно, например, в инструкции «К=К+1» 1 есть константа, или для удобства обозначать идентификаторами: *pi*=3,1415926536. Только значение *pi*нельзя изменить, так как это константа, а не переменная.

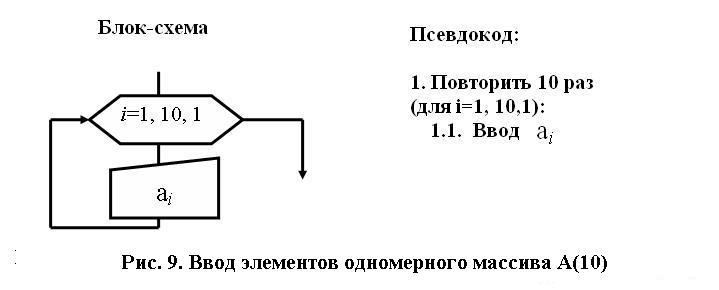
**Структурированные данные и алгоритмы их обработки**

          Для повышения производительности и качества работы необходимо иметь данные, максимально приближенные к реальным аналогам. Тип данных, позволяющий хранить вместе под одним именем несколько переменных, называется *структурированным.*Каждый язык программирования имеет свои структурированные типы. Рассмотрим структуру, объединяющую элементы одного типа данных, - *массив.*

*Массивом*называется упорядоченная совокупность однотипных величин, имеющих общее имя, элементы которой адресуются (различаются) порядковыми номерами (индексами). В качестве иллюстрации можно представить шкаф, содержащий множество пронумерованных ящиков (совокупность - «Ящик № 1», «Ящик № 2», «Ящик № 3» и т.д.; «Ящик» - общее имя всех ее элементов). Доступ к содержимому конкретного ящика (элементу массива) осуществляется после выбора ящика по его номеру (индексу). Элементы массива в памяти компьютера хранятся по соседству, одиночные элементы простого типа такого расположения данных в памяти не предполагают. Массивы различаются количеством индексов, определяющих их элементы.

*Одномерный массив*(шкаф ящиков в один ряд) предполагает наличие у каждого элемента только одного индекса. Примерами одномерных массивов служат арифметическая *(аi)*и геометрическая *(bi)*последовательности, определяющие конечные ряды чисел. Количество элементов массива называют *размерностью.*При определении одномерного массива его размерность записывается в круглых скобках, рядом с его именем. Например, если сказано: «задан массив A (10)», это означает, что даны элементы: *a 1, a 2, …, a 10*. Рассмотрим алгоритмы обработки элементов одномерных массивов.

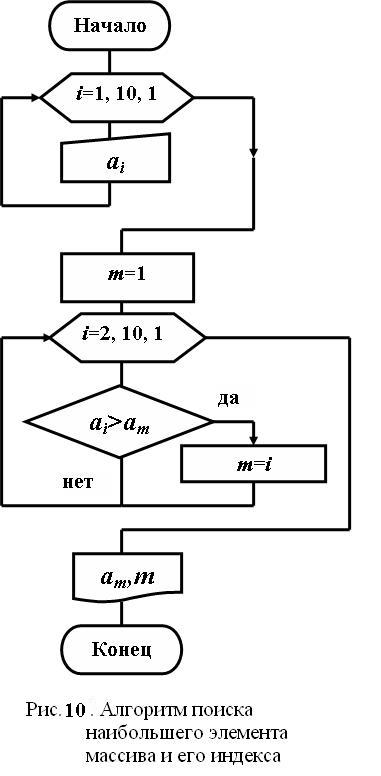
           Ввод элементов одномерного массива осуществляется поэлементно, в порядке, необходимом для решения конкретной задачи. Обычно, когда требуется ввести весь массив, порядок ввода элементов не важен, и элементы вводятся в порядке возрастания их индексов. Алгоритм ввода элементов массива А(10) представлен на рис.9.



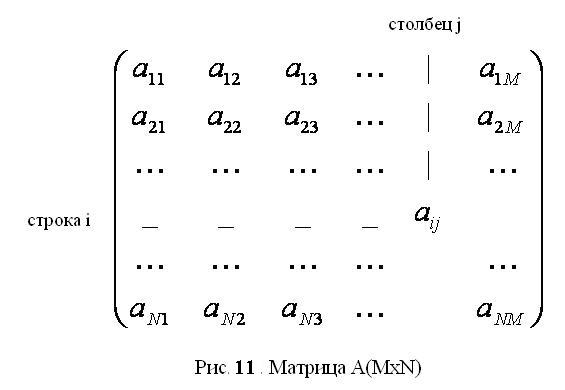
               Пример 4.

          В заданном числовом массиве A(l0) найти наибольший элемент и его индекс, при условии, что такой элемент в массиве существует, и единственный.

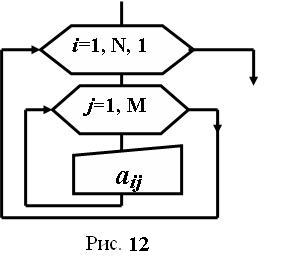
          Обозначим индекс наибольшего элемента *т.*Будем считать, что первый элемент массива является наибольшим *(т*= 1). Сравним поочередно наибольший с остальными элементами массива. Если оказывается, что текущий элемент массива *а* *i*(тот, c которым идет сравнение) больше выбранного нами наибольшего *ат,*то считаем его наибольшим *(т*=*i)*(рис.10).



         Рассмотрим *двумерный массив*(шкаф с множеством ящиков, положение которых определяется двумя координатами – по горизонтали и по вертикали). В математике двумерный массив (таблица чисел) называется матрицей. Каждый ее элемент имеет два индекса *а* *ij,*первый индекс *i*определяет номер строки, в которой находится элемент (координата по горизонтали), а второй *j –*номер столбца (координата по вертикали). Двумерный массив характеризуется двумя размерностями N и М, определяющими число строк и столбцов соответственно (рис. 11).



          Ввод элементов двумерного массива осуществляется построчно, в свою очередь, ввод каждой строки производится поэлементно, тем самым определяется циклическая конструкция, реализующая вложение циклов. Внешний цикл определяет номер вводимой строки ( *i*), внутренний – номер элемента по столбцу ( *j*). На рис. 12 представлен алгоритм ввода матрицы A(MxN) .



            Пример 5.

        Задана матрица символов (100х100), представляющая собой карту ночного неба; звездам на карте соответствует символы «\*». Определить: сколько звезд на карте?

          Алгоритм решения задачи достаточно прост, необходимо перебрать все элементы матрицы и посчитать, сколько среди них символов «\*». Обозначим К переменную – счетчик. На рис 13. представлена блок-схема решения этой задачи.

