РАЗРАБОТКА АЛГОРИТМОВ

Алгоритм – точный набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи.

Алгоритм можно описать:

– обычным текстом или математическими формулами (как, например, алгоритм поиска корней квадратного уравнения);

– кодом программы (для программистов это обычный и основной способ обмена алгоритмами, так как их можно просто описать в обычном текстовом редакторе или редакторе коды и также быстро можно исправить, при необходимости);

– блок-схемой алгоритмы (изображает последовательность действий стандартными графическими блоками).

Для упрощения построения блок-схем алгоритмов используют специализированные векторные редакторы – Microsoft Visio, Microsoft Word и т.п. Для стандартизации блок-схем используется ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения».

Перечислим наиболее часто используемые блоки (рис.1):

Рис.1. Основные блоки алгоритмов.

Для составления целого алгоритма отдельные блоки соединяются между собой линиями, называемыми линиями потока (имеется ввиду поток изменяемых данных). Несколько блоков, объединённых между собой линиями потока образуют алгоритмическую структуру, которую можно отнести к одному из ниже приведённых видов. Подавляющее большинство алгоритмов строится на основе шести базовых алгоритмических структур:

– линейная;
– ветвление (полное, неполное);
– многоальтернативное ветвление;
– параметрический цикл;
– цикл с предусловием;
– цикл с постусловием.

Линейный алгоритм используется в том случае, когда выполнение задачи не зависит от данных или условий, например, вычисление площади круга (рис.2).

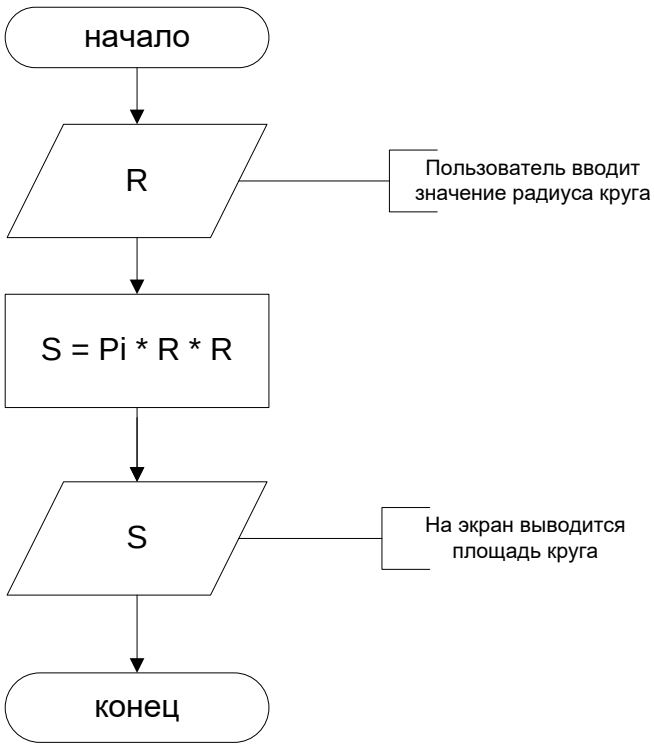
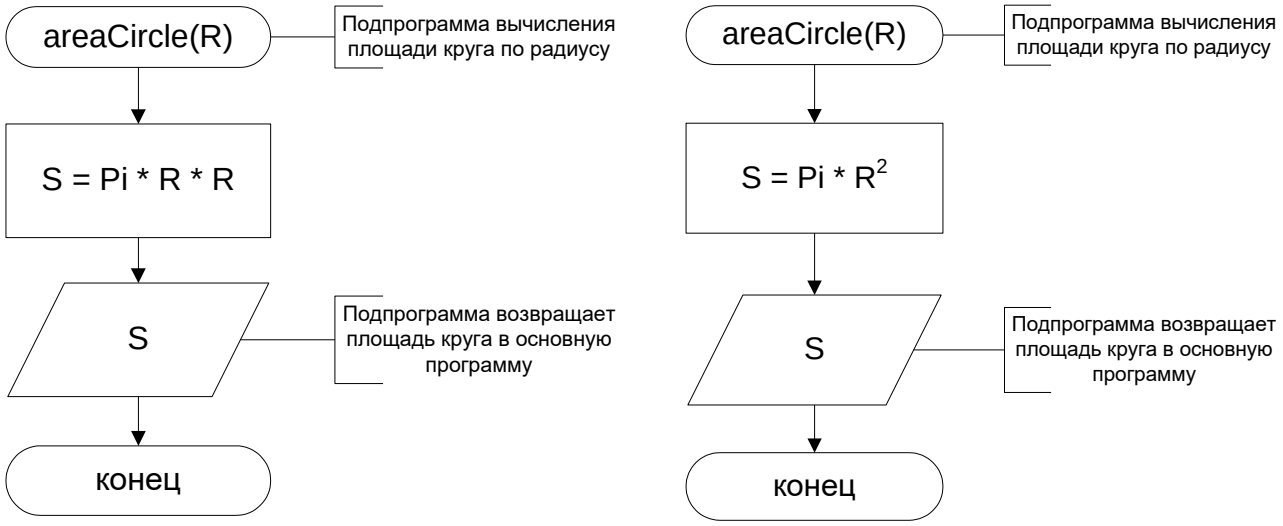


Рис.2. Линейный алгоритм.

Обратите внимание, что для пояснения сущности производимых действий можно использовать комментарии к блокам алгоритма (рис.2, справа).

Для реализации часто встречающихся задач, обычно, используют подпрограммы, например подпрограмма вычисления степени числа. Если нужно возвести число в степень 2 ещё можно написать R*R как и было показано в алгоритме выше без вызова подпрограммы, но что если нужно будет возводить в нецелую или отрицательную степень или степень числа будет большой? В этих случая как раз удобно вызвать подпрограмму по имени – она взамен вернёт вычисленное значение или выполнит иное необходимое действие. Скажем, в языке JavaScript для вычисления степени числа используется функция pow из библиотеки Math (математика), вызов функции записывается так: Math.pow(x, y) – это означает вычислить и вернуть x, возведенное в степень y. Эта функция стандартная и включена в сам язык программирования, но можно и самому создавать подобного рода подпрограммы, например, функцию, вычисляющую площадь круга по значению радиуса (рис.3а).



а) б)

Рис.3. Алгоритм функции площади круга.

В данный алгоритм добавим использование стандартной функции возведения в степень, но не обязательно использовать синтаксис

конкретного языка программирования – Math.pow(R, 2). Достаточно будет описать привычными символами математики (рис.3б).

Итак, мы описали свою собственную подпрограмму, теперь рассмотрим порядок её использования в основной алгоритме (рис.4).

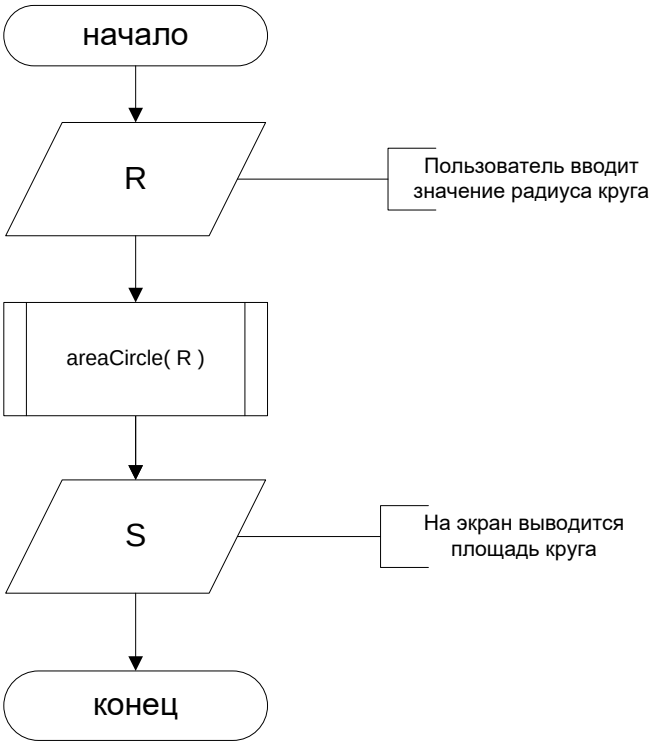


Рис.4. Алгоритм вычисления площади круга.

Уточним ещё один немаловажный момент, если окажется так, что блок-схема алгоритма не помещается на один лист или, будучи размещенной в рамках одного листа, на схеме окажется слишком много запутанных соединяющих линий, то можно разрывать алгоритм с помощью специальных «соединителей», назначая им порядковые номера (рис.6).

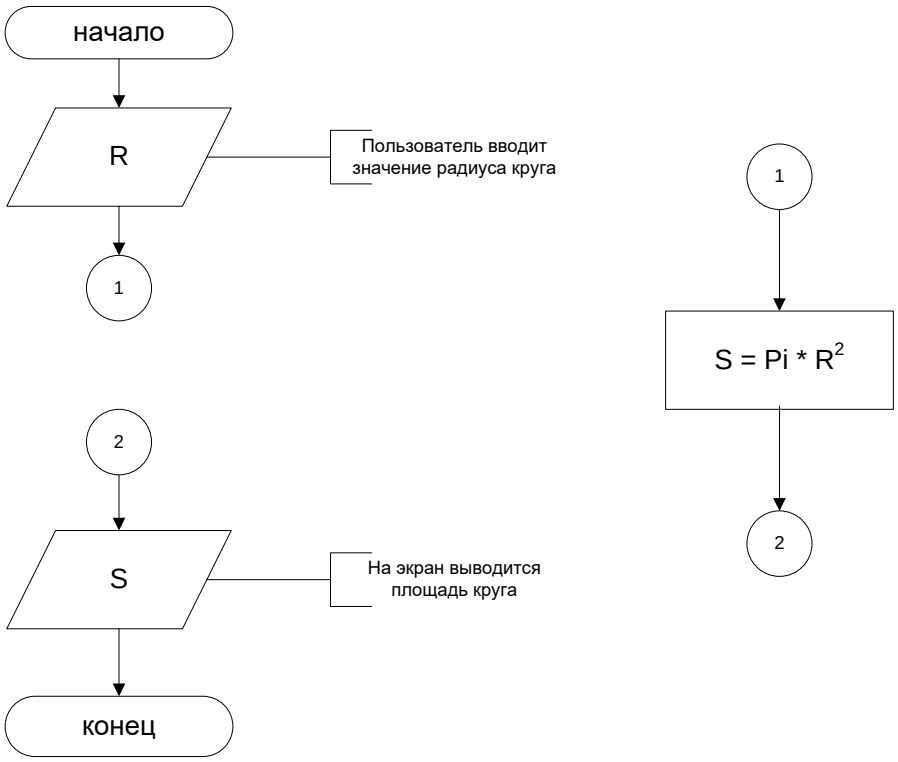


Рис.5. Использование соединителей внутри листа.

Алгоритмическая структура ветвления используется когда процесс решения задачи зависит от данных или иных условий, например, при решении квадратного уравнения (зависимость от дискриминанта). Предположим, есть два квадратных поля, известны стороны квадратов, стоит задача найти площадь максимального поля. При составлении данного алгоритма стоит учесть, что нет смысла считать площадь обоих полей, зазря растрачивая ресурсы компьютера, более рационально сначала выяснить б***о***льшую из двух сторону квадрата и уже затем вычислить только одну искомую площадь (рис.6).

Обратите внимание, что линии потока, идущие «слева-направо» и «справо-налево» и сходящиеся в один поток после блока ветвления, не должны смыкаться в одной точке (рис.6, в центре), что порождало бы неоднозначную ситуацию (например, пересечение одной линии потока – горизонтальной, другой линии – вертикальной).

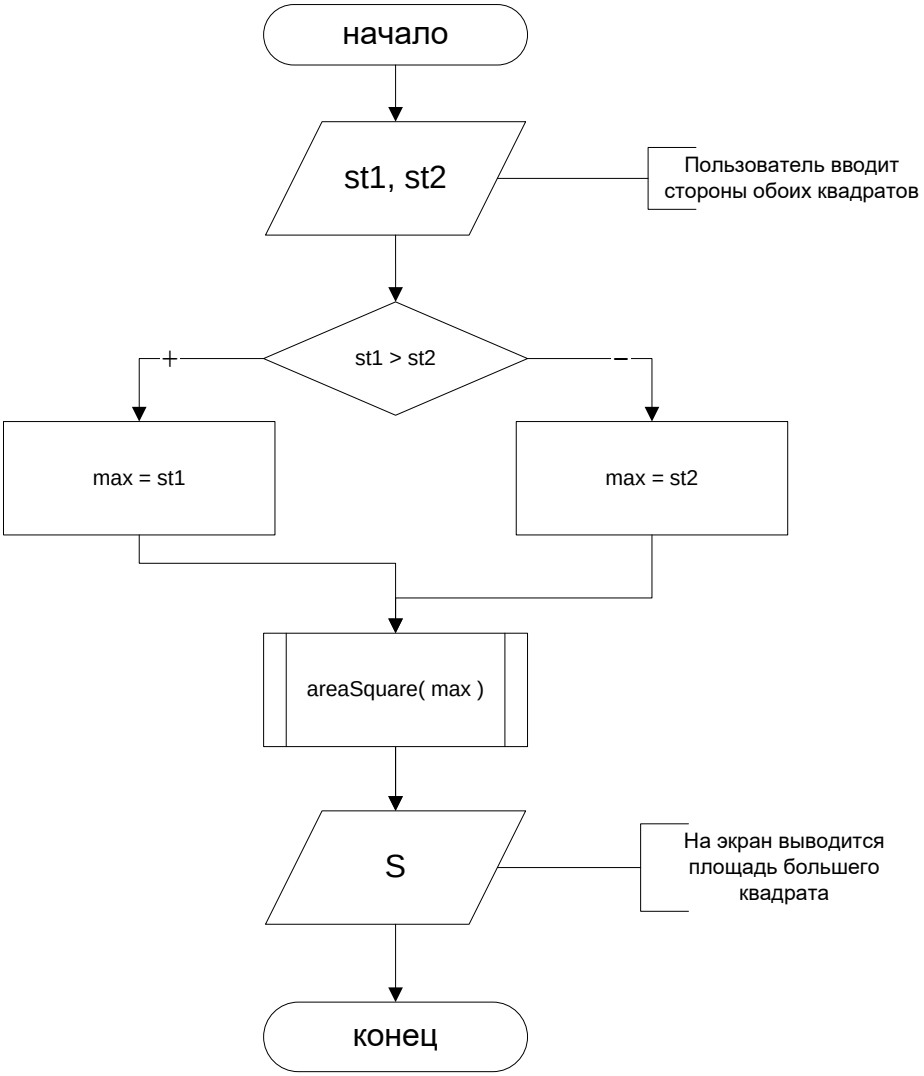


Рис.6. Алгоритм с полным ветвлением.

В блоке вида «ромбик» располагается проверяемое условие. Можно написать просто словами, например, «идёт дождь», при этом нет смысла ставить знак вопроса, так как блок и подразумевает, что внутри него располагается вопрос. А вот обозначить направления потока следует, то есть куда двигаться при истинности и ложности проверяемого условия. Направления потока после проверки условия можно обозначить как парой («+», «–»), так и парой («да», «нет»).

Под неполным ветвлением понимают ситуацию когда только одна ветвь алгоритмической структуры ветвления нагружена действием, например, если пошёл дождь, то укройся под зонтиком, иначе не предпринимай никаких действий (рис.7).

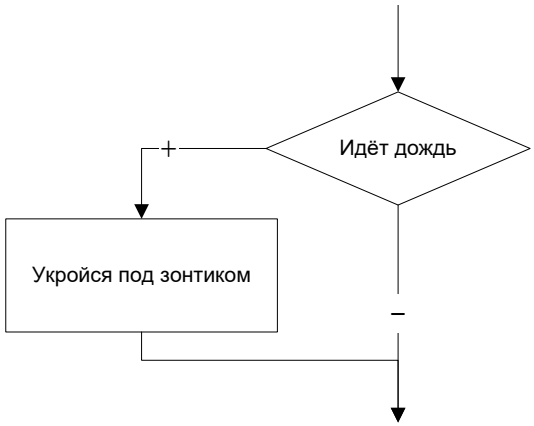


Рис.7. Неполное ветвление.

Зачастую возникают ситуации, когда после проверки условия возможны несколько выходов (более двух), тогда используется третья алгоритмическая структура – многоальтернативное ветвление. Рассмотрим пример алгоритма (рис.8) перевода числового представления оценки ученика в словесное (2 – неудовлетворительно, 3 – удовлетворительно, 4 – хорошо, 5 – отлично).

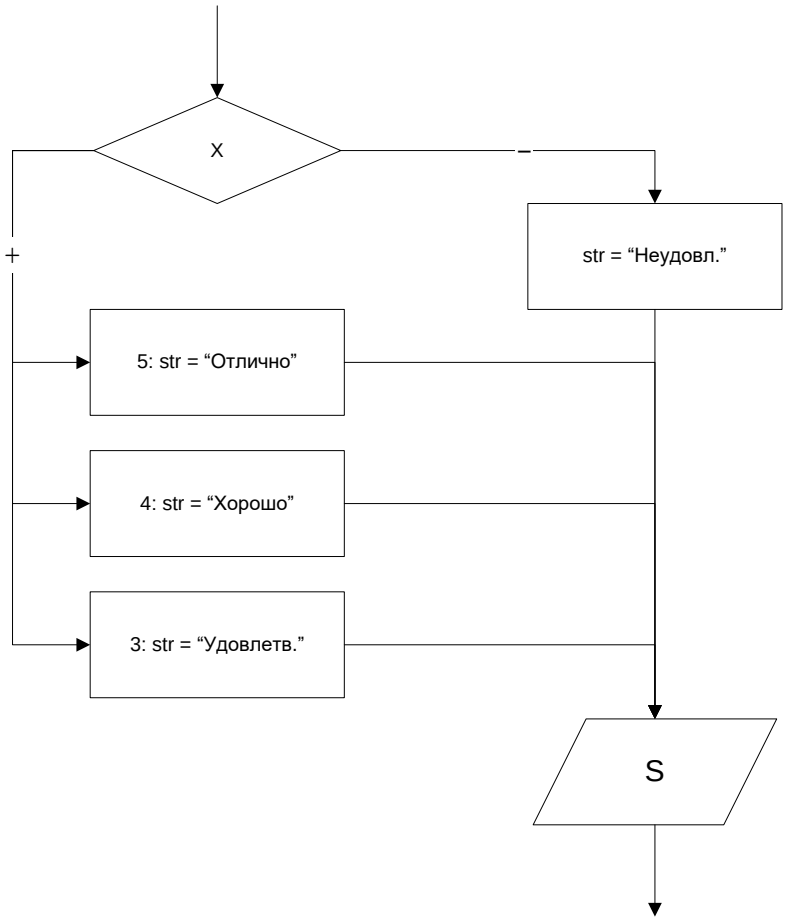


Рис.8. Алгоритмическая структура – многоальтернативное ветвление.

Обратите внимание, что в блоке проверки условия («ромбик») пишется, в данном случае, просто имя проверяемой переменной, а в самих блоках действия её значение и действие в этом случае. Если требуется указать диапазон значений вместо одного, то можно написать так: `10..20`, если же требуется просто перечислить некоторые значения, то можно их указать так: `2, 4, 6, 16`. Если же ни в одном из вниз идущих путей не было найдено значение переменной, указанной в блоке проверки условия, то можно использовать путь «иначе», тот, который обозначен линией потока с символом «–» (этот путь не является обязательным для изображения на алгоритме и используется факультативно).

Одной из важных алгоритмических структур является цикл, он предназначен для организации работы с повторяющимися действиями. Для его использования назначены три вида циклических алгоритмических структур – параметрический цикл, цикл с предусловием и цикл с постусловием.

Параметрический цикл назван так, потому что в шапке цикла должен присутствовать некий параметр, изменяющийся на каждой итерации цикла. В шапке цикла также должны быть определены начальное значение параметра, шаг его изменения и пределы его изменения. Рассмотрим алгоритм вычисления суммы чисел от 0 до N (рис.9).

Обратите внимание, что если шаг изменения параметра цикла равен +1, то его не следует обозначать в шапке цикла.

Структура параметрического цикла представлена двумя самостоятельными частыми: собственно шапка цикла и тело цикла. Шапка цикла изображена фигурой – «шестиугольник», тело цикла – это один или несколько других блоков, следующих за шапкой цикла и связанных замкнутой линией потока (та, которая возвращается обратно к шапке цикла).

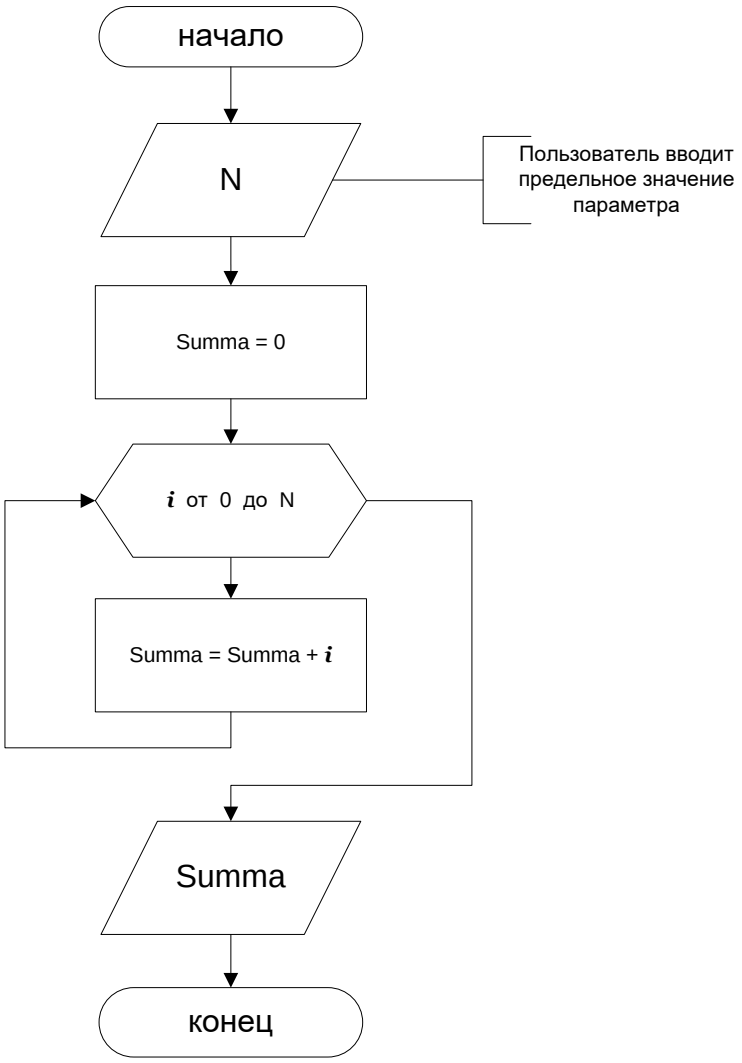


Рис.9. Алгоритм с параметрическим циклом.

Не всегда удобно использовать именно параметрический цикл, так как бывают задачи, в которых заранее не указать количество повторов. Предположим нужно найти минимальное число большее 99, которое нацело делится на N, например, на 17 (рис.10). В данном случае подходит цикл с предварительной проверкой условия, то есть до выполнения тела цикла – сначала проверяем условие, а потом выполняем тело цикла. Обратите внимание, что в данном случае в блоке проверки условия использовался синтаксис языка программирования Pascal, что не всегда удобно, так как читающие алгоритм могут не знать именно этого языка программирования. В блоке используется операция вычисления остатка от целочисленного деления mod, которая возвращает результат «0», если делимое делится на делитель нацело. Именно поэтому в блоке проверки условия стоит сравнение с

нулём. Обычно эту операцию используют при проверке числа на чётность/нечётность, так как при целочисленном делении проверяемого числа на два, возможны всего два ответа «0» – при чётном числе или «1» – при нечётном. Например, 15 mod 2 = 1, 16 mod 2 = 0.

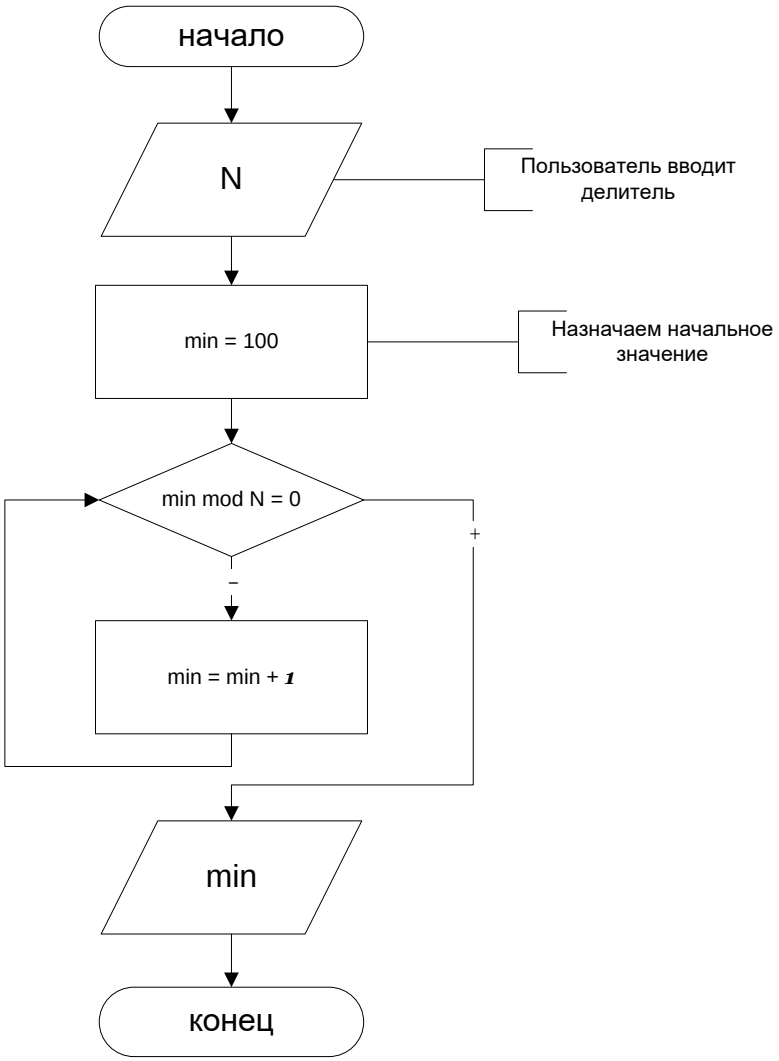


Рис.10. Использование цикла с предусловием.

В качестве заключительной алгоритмической структуры рассмотрим цикл с постусловием – он необходим в тех случаях, когда тело цикла должно выполниться хотя бы один раз. Одним из распространённых примеров может случить организация «паузы» (задержки экрана) после выполнения программы, для чего опрашивают клавиатуру до тех пор, пока пользователь не нажмёт заданную клавишу, например, «Escape».

Пусть в нашем алгоритме будет заложена такая задача: пользователь вводит любое число, программа в ответ выводит его квадрат

(рис.11) и, затем, если число было равно нулю, то нужно покинуть цикл и завершить программу, а иначе продолжать вводить число и выводить его квадрат.

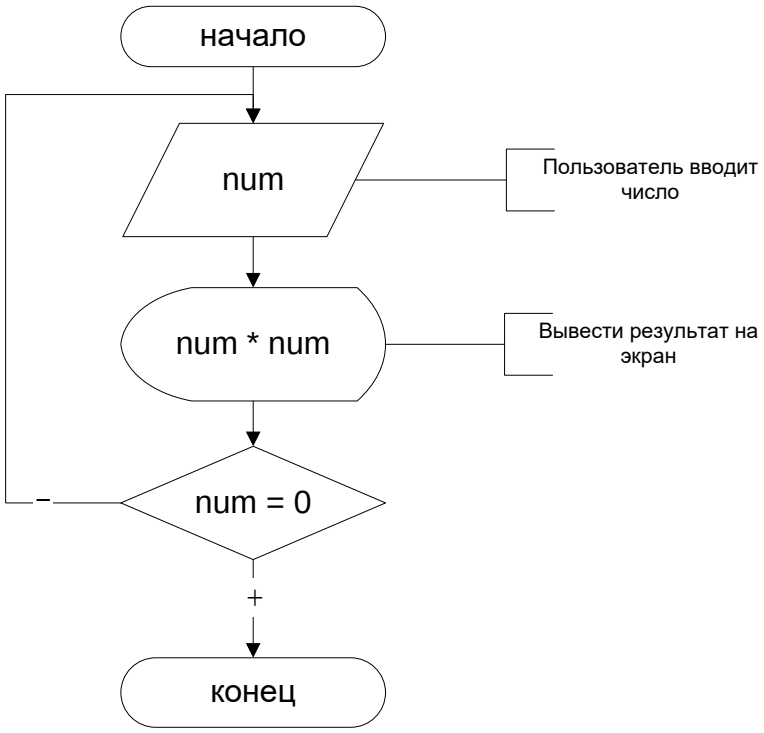
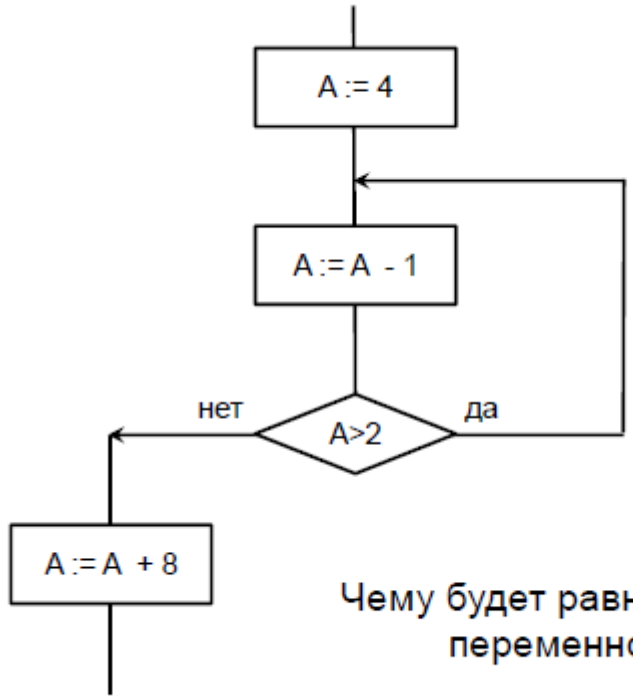


Рис.11. Использование цикла с постусловием.

Итак, мы изучили все основные алгоритмические структуры. Пришло время проверить себя, ниже будут приведены несколько участков алгоритмов, необходимо будет разобраться в их работе и ответить на поставленный вопрос.

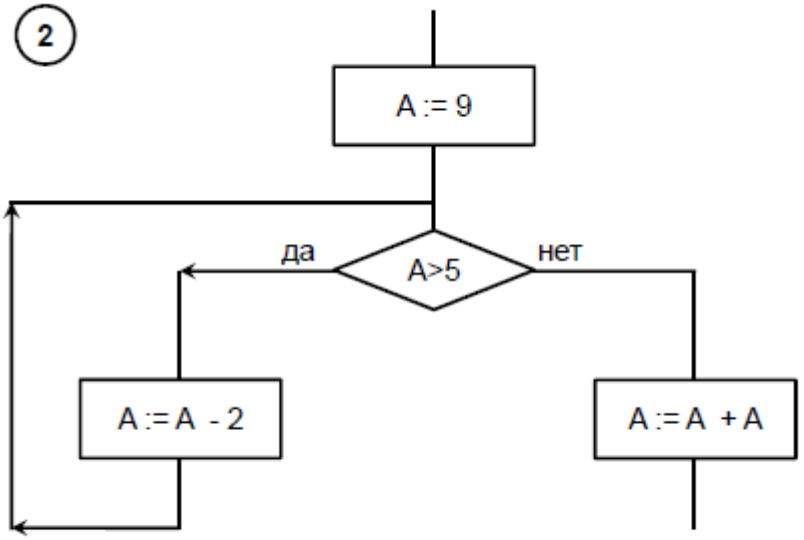
Задания для самоконтроля

1



Чему будет равно значение переменной A ?

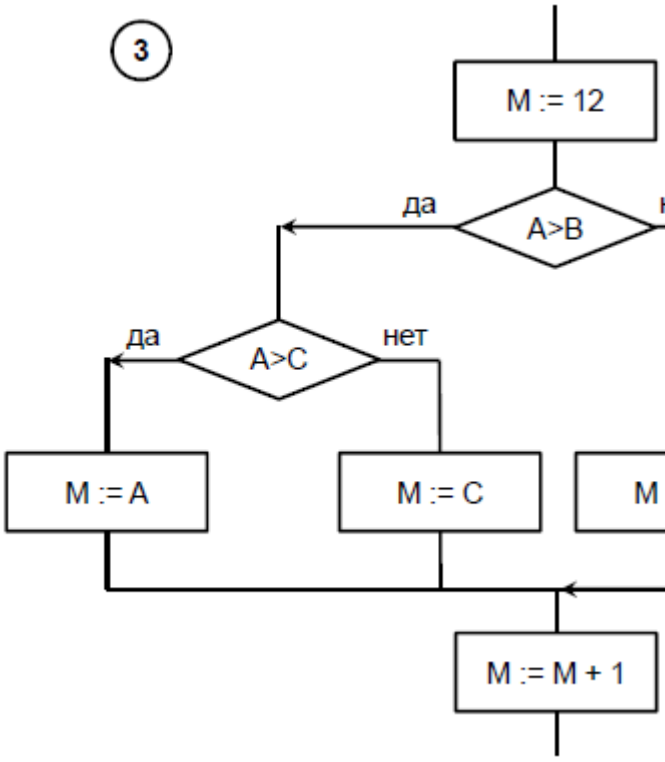
2



Чему будет равно значение переменной A ?

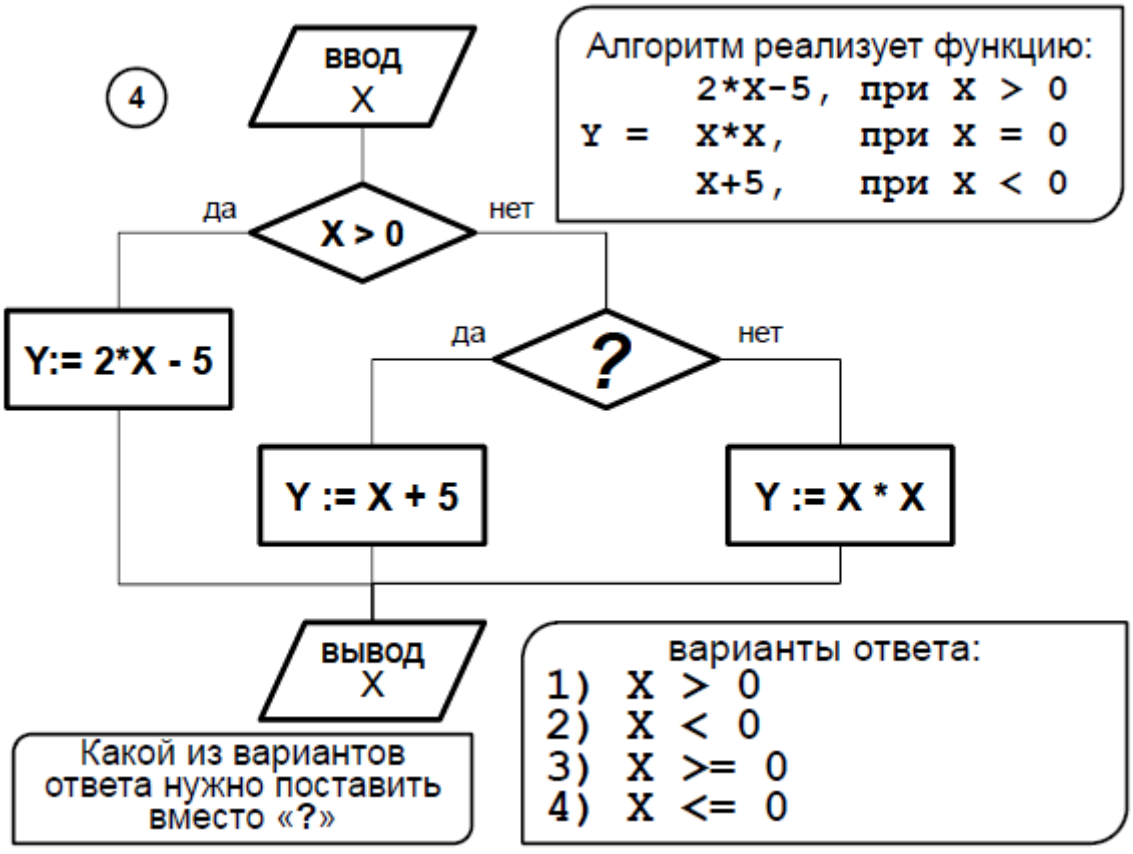
3

до выполнения алгоритма значения равны:
A = 15, B = 13, C = 15



Чему будет равно значение переменной M ?

4



Алгоритм реализует функцию:

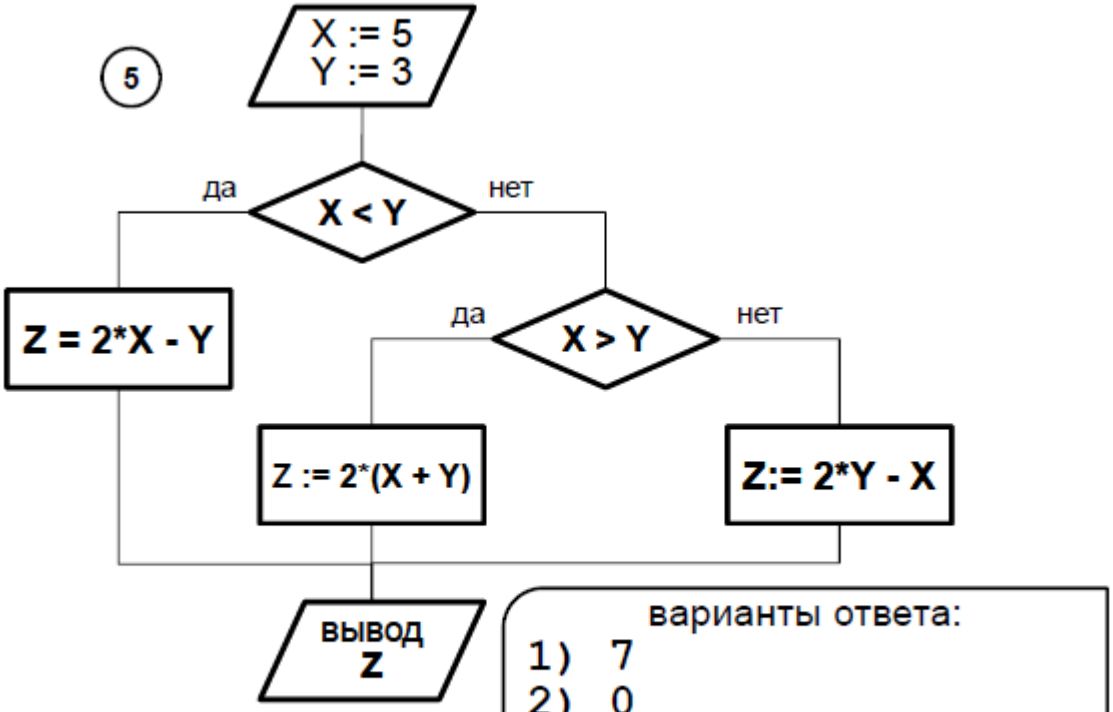
$$Y = \begin{cases} 2*X-5, & \text{при } X > 0 \\ X*X, & \text{при } X = 0 \\ X+5, & \text{при } X < 0 \end{cases}$$

Какой из вариантов ответа нужно поставить вместо «?»

варианты ответа:

1) X > 0
2) X < 0
3) X >= 0
4) X <= 0

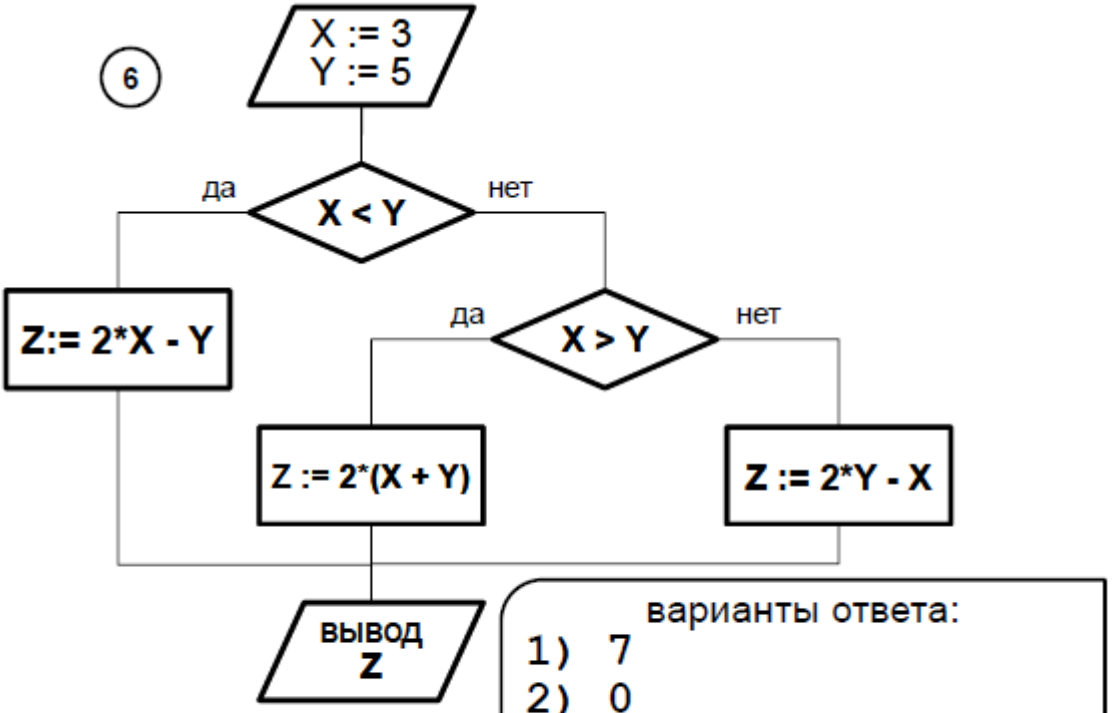
5



варианты ответа:

1) 7
2) 0
3) 16
4) 1

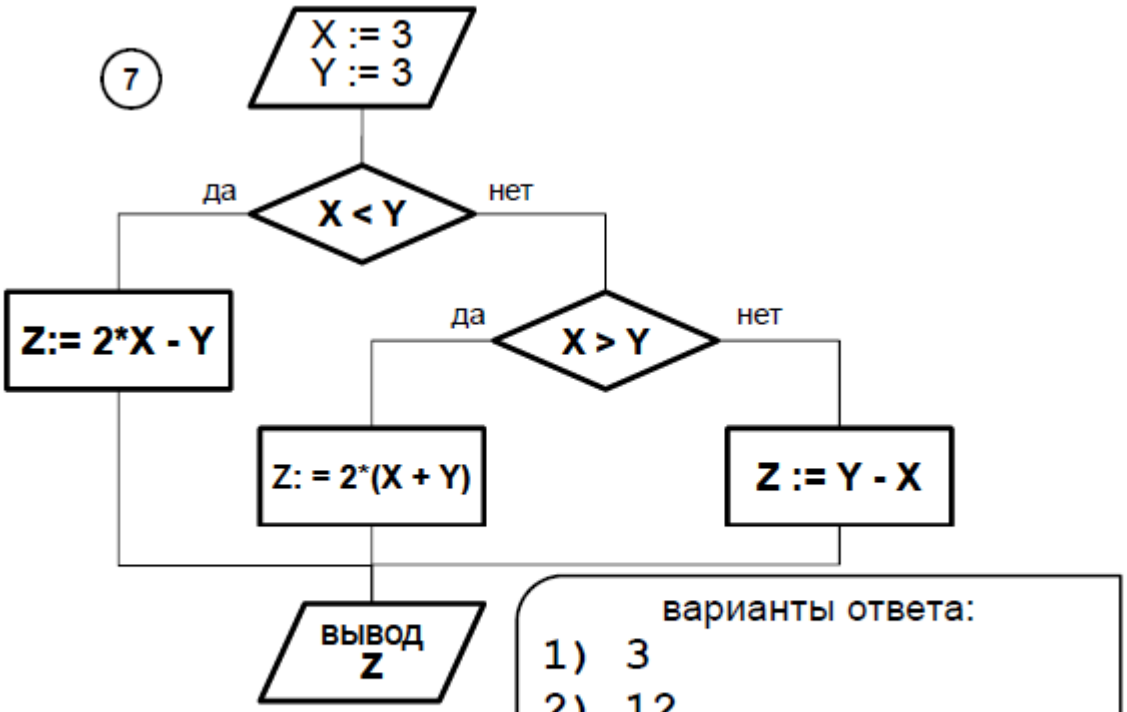
6



варианты ответа:

1) 7
2) 0
3) 16
4) 1

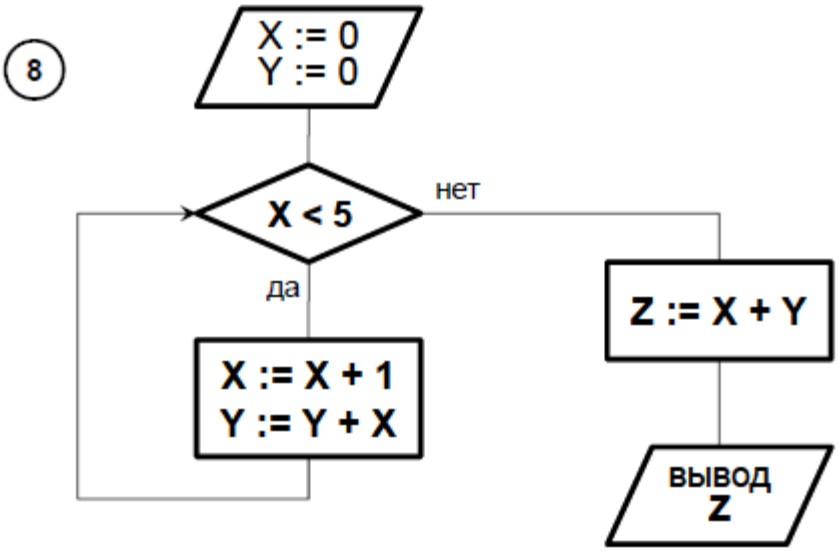
7



варианты ответа:

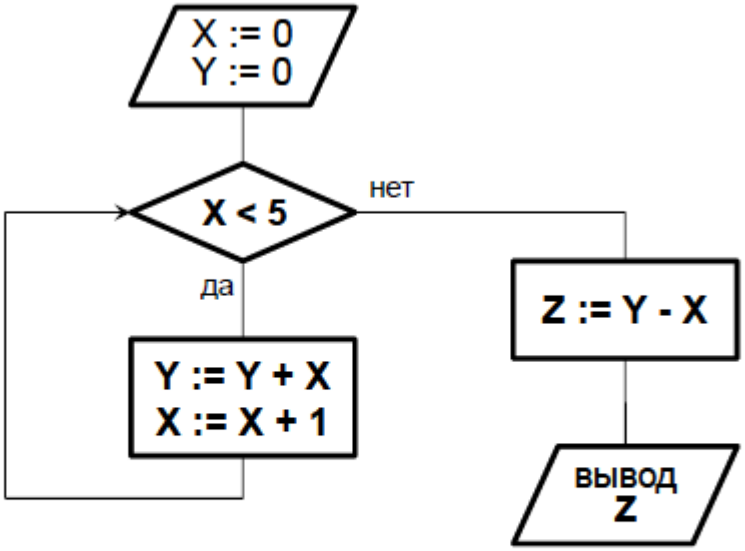
1) 3
2) 12
3) 0
4) 5

8



Z = ?

9



Z = ?

10

Исполнитель способен выполнять только две команды:

1) прибавить 1
2) прибавить 4

Сколько различных чисел может получить исполнитель из числа 2 с помощью программы, которая содержит не более трех команд (0..3).

Исходное число тоже учитывать.

Задания для самостоятельного исполнения

Задание №1 для самостоятельного исполнения

Составьте схему алгоритма

Вычисления корней квадратного уравнения…

$$a*x^2 + b*x + c = 0$$

$$x_{1,2} = \frac{-b \pm D}{2a}$$

где $D = \sqrt{b^2 - 4ac}$

- схему алгоритма исполнить в редакторе Word

Задание №2 для самостоятельного исполнения

Составьте схему алгоритма

Вычисления корней квадратного уравнения…

$$a*x^2 + b*x + c = 0$$

- составить алгоритм с учетом того, что возможны три случая для дискриминанта:

1) если D > 0, то имеются два различных вещественных корня, которые можно вычислить по следующим формулам:

$$x_{1,2} = \frac{-b \pm D}{2a}$$

2) если D = 0, то имеется единственный корень: x = -b/2a.

3) если D < 0, то вещественных корней нет.

- схему алгоритма исполнить в редакторе Word

Задание №3 для самостоятельного исполнения

Составьте схему алгоритма

Составить алгоритм расчета транспортного налога для физических лиц в зависимости от налоговой ставки на 1 л/с мощности двигателя:

– до 100 л.с. – 2,5 р.
– до 150 л.с. – 3,5 р.
– до 200 л.с. – 5,0 р.
– до 250 л.с. – 7,5 р.
– свыше 250 л.с. – 15 р.

Пользователь вводит значение мощности двигателя, а алгоритм вычисляет транспортный налог и выводит на экран.

- исполнить в редакторе Word в двух вариантах:
1) только с использованием блоков ветвлений
2) с использованием блока многоальтернативного выбора

Задание №4 для самостоятельного исполнения

Составьте схему алгоритма

Составить алгоритм вычисления квадратного корня Z из произвольного положительного вещественного числа X методом Герона:

$$Z_n = (Z_{n-1} + X/Z_{n-1})/2, \quad \text{где } Z_0=1$$

Чем больше n тем точнее значение корня, обычно достаточно и 5 повторений.

Пользователь вводит значение X и n, а алгоритм вычисляет приближенное значение корня Z и выводит его на экран.

Шаги работы алгоритма для числа X = 9:

$Z_0 = 1$

$Z_1 = (Z_0 + X/Z_0)/2 = (1 + 9/1)/2 = 5$

$Z_2 = (Z_1 + X/Z_1)/2 = (5 + 9/5)/2 = 3.4$

$Z_3 = (Z_2 + X/Z_2)/2 = (3.4 + 9/3.4)/2 = 3.02$

Задание №5 для самостоятельного исполнения

Составьте схему алгоритма

Составить алгоритм вычисления квадратного корня Z из произвольного положительного вещественного числа X методом Герона:

$$Z_n = (Z_{n-1} + X/Z_{n-1})/2, \quad \text{где } Z_0=1$$

Пользователь вводит значение X и q – точность вычисления, а алгоритм вычисляет приближенное значение корня Z и выводит его на экран.

Точность q считается достигнутой, если текущее значение Z_n отличается от предыдущего на величину не большую чем q.

Задание №6 для самостоятельного исполнения

Составьте схему алгоритма

Пусть задано натуральное число N. Пусть в S накапливается сумма натуральных чисел от 1 до N. Пусть задано ограничение на сумму S, назовем его MaxS. Необходимо составить схему алгоритма подсчета максимально возможного количества **K** натуральных чисел, входящих в сумму S, не превышающую MaxS.

Пример:

```
N=5
MaxS=8

_____________________________
S=1+2+3=6        K=3        S<=MaxS
S=1+2+3+4=10     K=4        S> MaxS

_____________________________
```

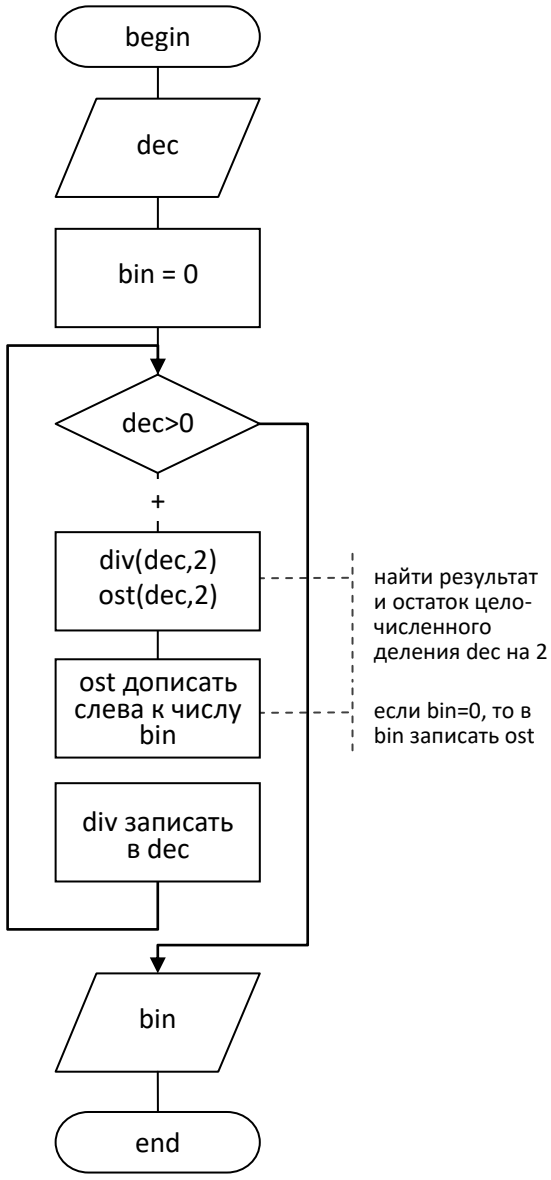
Ответ:

```
K=3
```

- схему алгоритма исполнить в редакторе Word

АЛГОРИТМЫ И ПРОГРАММНЫЙ КОД

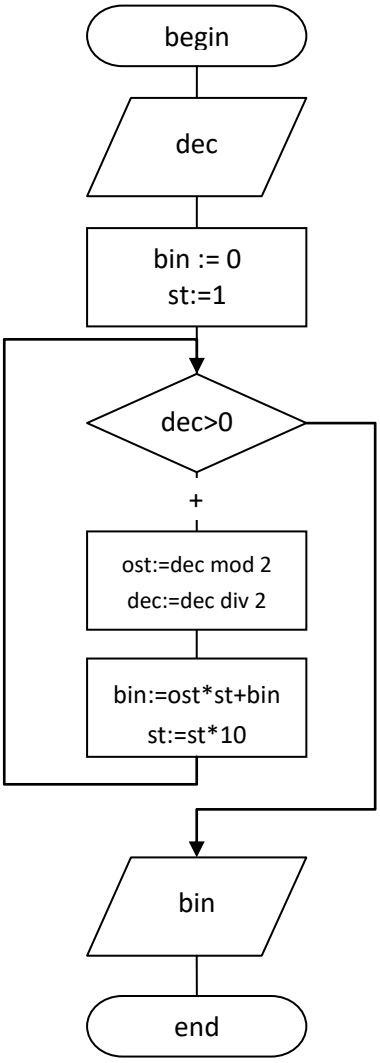
Алгоритмом определяется порядок действий для решения конкретной задачи. Для инициализации работы алгоритма нужны входные данные. Для алгоритма также заранее должны быть указаны и выходные переменные, приобретающие своё значение в процессе работы алгоритма. Схема алгоритма, обычно, представляет собой достаточно общее описание последовательности действий, не уточняющее мелкие детали реализации, и этого перечисления вполне достаточно для такого исполнителя как человек, так как его интеллект вполне позволяет додумать ему очевидные недосказанности. Однако для компьютера такая недосказанность может стать «камнем преткновения», поэтому компьютерная программа, даже написанная на языке высокого уровня, не вполне соответствует исходному словесному описанию, более детализирована частностями и содержит недостающие детали, выраженные синтаксисом языка и семантикой операторов.



Для определенности рассмотрим пример перевода десятичного числа 13 в двоичную систему счисления в виде таблицы, демонстрирующей итерации решения и последовательный сбор ответа в ***обратном порядке из пошаговых результатов***, и схему алгоритм по переводу целого десятичного числа в двоичную систему счисления:

исходное значение = **13**			
шаг	значение числа	целочисленное деление на 2	остаток от деления
1	13	6	1
2	6	3	0
3	3	1	1
4	1	0	1
ответ = **01101** или просто **1101**			

Приведенная схема алгоритма имеет достаточно общий характер: не уточняются типы входных и выходных данных, не указаны методы работы с данными, не навязывается язык программирования синтаксическими вкраплениями. Тем не менее, зачастую схема алгоритма составляется с детализацией синтаксиса, методов и типов данных:



Вид надписей в блоках данной схемы явно указывает на язык Pascal, а содержание определенно отражает метод работы с данными и подчеркивает, что ответ формируется в целочисленном, а не в строковом виде.

Для большинства случаев это удобно, так как алгоритм практически представляет программу. Сравните сами:

```
var dec,bin,st,ost: word;
begin
  writeln('Введите число: ');
  read(dec);
  bin:=0; st:=1;
  while dec>0 do
  begin
    ost:=dec mod 2;
    dec:=dec div 2;
    bin:=ost*st+bin;
    st:=st*10;
  end;
  write(bin);
end.
```

Однако следует понимать, что алгоритм может быть использован для реализации на любом языке программирования, не только отличающегося синтаксисом, но и обладающего иными, возможно лучшими, методами работы с данными.

Для примера приведу реализацию на языке Python:

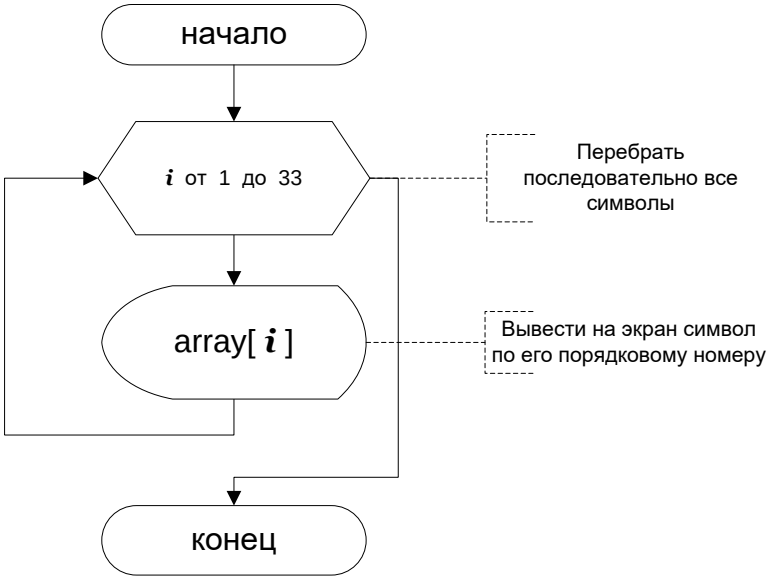
```
def decToBin(dec):
    bin=0                   # тут будем накапливать результат
    st=1                    # степень, чтобы цифру писать в определенный разряд
    while dec>0:
        bin+=dec%2*st       # остаток от целочисленного деления на 2
                            # умножить на степень и добавить к результату
        dec//=2             # dec целочисленно делить на 2 и записать в dec
        st*=10              # степень st домножить на 10 и записать результат в st
    return bin

dec = int(input('Введите число: '))
print(decToBin(dec))
```

Синтаксис Python ближе к C++ или JavaScript, но, несмотря на это, если у вас есть опыт написания программ на языках структурного программирования, то код Python'а вы тоже поймете и сможете адаптировать к своей среде разработки. Именно поэтому, в большинстве случаев, программисты для передачи логики работы обмениваются не схемами алгоритмов, а кусками программного кода. Тем не менее, схемы алгоритмов используют для документирования коммерческой работы, для наглядной передачи логики решения задачи и при обучении.

Обратите внимание, что основную работу без изменений логики обработки данных я вынес в отдельную функцию. Так делают для систематизации программы, сокращения повторяющихся участков кода и удобства использования разработанной части кода во всех языках программирования. Подобного рода функции, например, очень удобно использовать для расширения функциональных возможностей Excel средствами встроенного объектно-ориентированного языка программирования VBA (Visual Basic for Applications).

ОБРАБОТКА МАССИВОВ

До сих пор в алгоритмах мы рассматривали одиночные переменные, но в программах, ориентированных на решение практических задач, данные, как правило, хранят в определённых структурах, например, в массивах данных. Массив – это проиндексированная (то есть с порядковыми номерами) последовательность элементов одного типа (то есть только строки или только целые числа и т.п.). В частности таблица умножения – это такой двумерный массив, заполненный целыми числами по определённому правилу. Чтобы обратиться к определённому элементу из массива, нужно сначала написать имя массива, а, затем в скобках (иногда используют круглые скобки, иногда – прямоугольные) номер его позиции в массиве, например, так: m[5]. Если этот массив одномерный, то количество индексов – один, если двумерный, то два индекса (например, m[1, 3]) и т.д. Примером одномерного массива может служить алфавит, в котором под определённым порядковым номером расположен определённый символ. Во многих языках программирования в массивах индексация по умолчанию начинается с нуля, то есть первый элемент массива имеет порядковый номер равный нулю, второй элемент – 1 и т.д. В некоторых случаях можно изменить нижнюю границу массива и начать нумерацию с единицы. Пусть у нас для хранения алфавита русского языка назначена переменная array, индексация начинается с единицы, тогда элемент array[3] будет содержать символ «в». Разработаем алгоритм вывода на экран символов русского языка:



Обратите внимание, что несколько примеров по составлению алгоритмов обработки одномерных и двумерных массивов вы можете изучить в презентации по теме «Алгоритмизация» на страничке «Информатика» сайта pCoding.ru.

Для примера рассмотрим немного более трудоёмкую задачу: нужно разработать алгоритм и программу генерации массива из count (count задаёт пользователь) чисел (в диапазоне (0,10]) с последующим подсчётом суммы элементов массива больших 5. Можно сказать, что словесное описание алгоритма содержится в самой задаче. Оформим решение в двух шагах:

– на первом шаге пользователь вводит количество элементов массива, программа генерирует числа и заполняет ими массив;

– на втором шаге программа перечисляет все элементы, но для суммирования выбирает лишь те из них, которые соответствуют условию задачи, и суммирует их.

Для разработки алгоритма будем использовать специальный редактор Блок-схем-алгоритмов AFCE (*Algorithm Flowchart Editor*). В левом окне редактора располагаются инструменты (блоки), по центру – окно редактирования блок-схемы, справа – исходный код программы и окно помощи (с нижней части) (см. рис.12).

Программный код в правом окне генерируется автоматически, вы имеете возможность выбрать язык программирования в выпадающем списке.

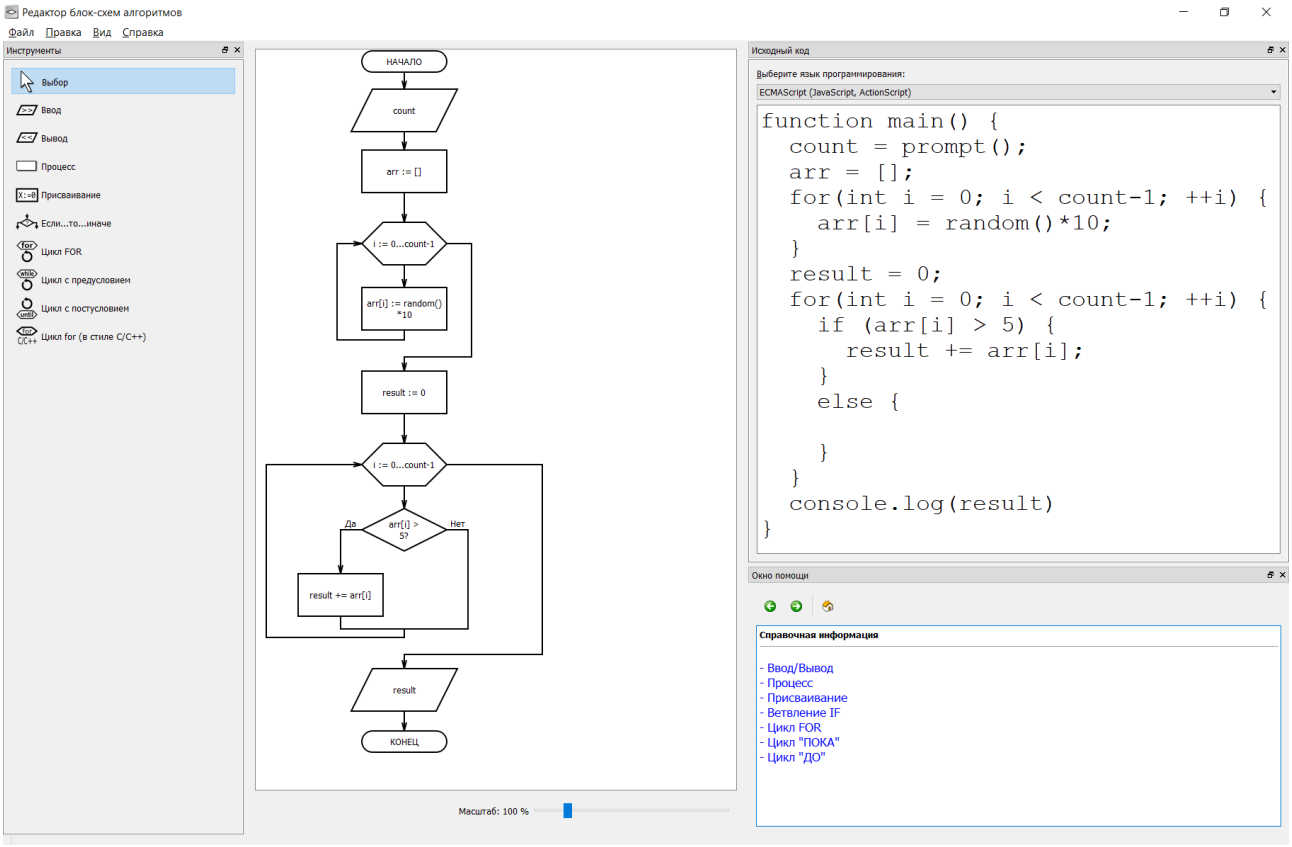


Рис.12. Окна редактора Algorithm Flowchart Editor.

Окна кода и блок-схемы масштабируются. Разработанный алгоритм можно сохранить как во внутреннем формате редактора для последующего редактирования, так и в виде графического файла через главное меню программы. Для того чтобы установить блок в схему алгоритма достаточно просто выделить его мышкой в левом окне и затем кликнуть в необходимую позицию в окне блок-схемы.

После разработки алгоритма сгенерированный код программы можно скопировать из окна кода, перенести в редактор, которым вы пользуетесь для отладки программ, и продолжить работу с программой уже без данного редактора. Однако, не всегда удаётся сгенерировать автоматически полностью корректный код. Иногда среда гене-

рации AFCE просто не учитывает необходимый программисту контекст использования, поэтому имеет смысл доработать программу.

Посмотрите доработки кода по обсуждаемой задаче и попробуйте самостоятельно разобраться во внесённых изменениях, при необходимости воспользуйтесь помощью интернета.

```
<!--
  Программа генерации массива
  из count чисел в диапазоне [0,10) // 10 не входит
  с последующим подсчётом
  суммы элементов массива больших 5.
-->
<script>
  var count = +prompt(); // вводим кол-во элементов
  var arr = []; // инициализируем массив
  for(let i = 0; i < count; ++i) {
    arr[i] = Math.random()*10; // заполняем элементы
  }
  var result = 0; // инициализируем переменную
  for(let i = 0; i < count; ++i) {
    if (arr[i] > 5) { // проверяем условие
      result += arr[i]; // суммируем
    }
  }
  document.write(result.toFixed(2)); // выводим результат
</script>
```

Несколько заданий для проверки усвоенного материала.

1. Разработайте программу, которая может проверить наличие/отсутствие определённого элемента в массиве.

2. Разработайте программу циклического сдвига элементов массива вправо (или влево). При циклическом сдвиге вправо все элементы массива сдвигаются на одну позицию вправо, а самый крайний справа уходит на позицию самого левого.

3. Пусть даны массивы arrA и arrB. Элементы массивов это целые числа в диапазоне от 0 до 100, массивы содержат разное количество элементов (от 0 до 10). Некоторые элементы в массивах могут быть одинаковые. Разработайте программу, которая из двух массивов делает один содержащий только неповторяющиеся элементы.

ВНИМАНИЕ!

Для заочной формы обучения по данной теме необходимо выполнить **контрольную работу** (см. на сайте ***pCoding.ru*** в разделе «Информатика») – разработать несколько алгоритмов и программ к ним