ИНФОРМАТИКА

7. Алгоритмизация и программирование

1. Понятие алгоритма

2. Свойства алгоритма

3. Формы записи алгоритмов

4. Базовые алгоритмические структуры

1. Понятие алгоритма

Алгоритм — заранее заданное понятное и точное предписание возможному исполнителю совершить определенную последовательность действий для получения решения задачи за конечное число шагов.

***Алгоритм* – это точно определенное описание способа решения задачи в виде конечной последовательности действий**.

Для представления алгоритма в виде, понятном ПК, служат языки программирования, с помощью которых пишется программа. Затем программа с помощью транслятора либо переводится в машинный код, либо исполняется.

Программа – упорядоченная последовательность команд, необходимых для управления компьютером (ПК).

Эти команды поступают на процессор как совокупность нулей и единиц, т.е. числами. Последовательность чисел – *машинный код.*

Языки программирования – это искусственные языки с ограниченным числом слов, значения которых понятно транслятору, и очень строгими правилами записи *команд* (*операторов*).

При нарушении формы записи программы возникают синтаксические либо логические ошибки. Поиск ошибок – ***тестирование***, процесс устранения ошибок – ***отладка***.

С помощью языков программирования создается ***текст программы.*** Чтобы получить работающую программу необходимо либо сразу перевести текст программы в машинный код (откомпилировать), либо сразу выполнять команды языка с помощью ***интерпретатора,*** который поочередно анализирует отдельные команды и затем сразу же выполняет их. После того как текущий оператор выполнен, интерпретатор перейдет к следующему. Такие программы работают медленно и не могут выполняться сами, отдельно от интерпретатора.

Компиляторы же полностью обрабатывают текст программы, просматривают его в поисках синтаксических ошибок и автоматически переводят его в машинный код. В результате получается компактная, быстрая «исполняемая» программа. Однако компиляторы неэффективны при работе с данными сложной структуры.

Этапы решения задач с помощью компьютера

Решение задач с помощью компьютера включает в себя следующие основные этапы, часть из которых осуществляется без участия компьютера.

1. **Постановка задачи:**

✔ сбор информации о задаче;
✔ формулировка условия задачи;
✔ определение конечных целей решения задачи;
✔ определение формы выдачи результатов;
✔ описание данных (их типов, диапазонов величин, структуры и т.п.).

Этапы решения задач с помощью компьютера

2. **Анализ и исследование задачи, модели:**

- анализ существующих аналогов;
- анализ технических и программных средств;
- разработка **математической модели**;
- разработка структур данных.

3. **Разработка алгоритма:**

- выбор метода проектирования алгоритма;
- выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
- выбор **тестов** и метода тестирования;
- проектирование алгоритма.

Этапы решения задач с помощью компьютера

4. Программирование:

- выбор языка программирования;
- уточнение способов организации данных;
- запись алгоритма на выбранном языке программирования.

5. Тестирование и отладка:

- синтаксическая отладка;
- отладка семантики и логической структуры;
- тестовые расчеты и анализ результатов тестирования;
- совершенствование программы.

Отладка программы — это процесс поиска и устранения ошибок в программе, производимый по результатам её прогона на компьютере.

Тестирование (англ. test — испытание) — это испытание, проверка правильности работы программы в целом, либо её составных частей.

Этапы решения задач с помощью компьютера

6. Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2 — 5.

7. Сопровождение программы: это работы, связанные с обслуживанием программ в процессе их эксплуатации

- доработка программы для решения конкретных задач;
- составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

2. Свойства алгоритма

1. **Понятность для исполнителя** — исполнитель алгоритма должен понимать, как его выполнять.
2. **Дискретность (прерывность, раздельность)** — алгоритм должен представлять решение задачи как последовательное выполнение простых (или ранее определённых) шагов (этапов).
3. **Определённость** — каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола.
4. **Результативность (или конечность)** — за конечное число шагов алгоритм либо должен приводить к решению задачи, либо останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо неограниченно продолжаться в течение заданного времени с выдачей промежуточных результатов.
5. **Массовость** — алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для класса задач, различающихся лишь исходными данными.

3. Формы записи алгоритма

- **Словесная** (запись на естественном языке);
- **Графическая** (изображения из графических символов);
- **Псевдокоды** (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- **Программная** (тексты на языках программирования).

Словесная форма записи алгоритма

Алгоритм в данной форме записи представляет собой последовательность действий:

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 шага 2.

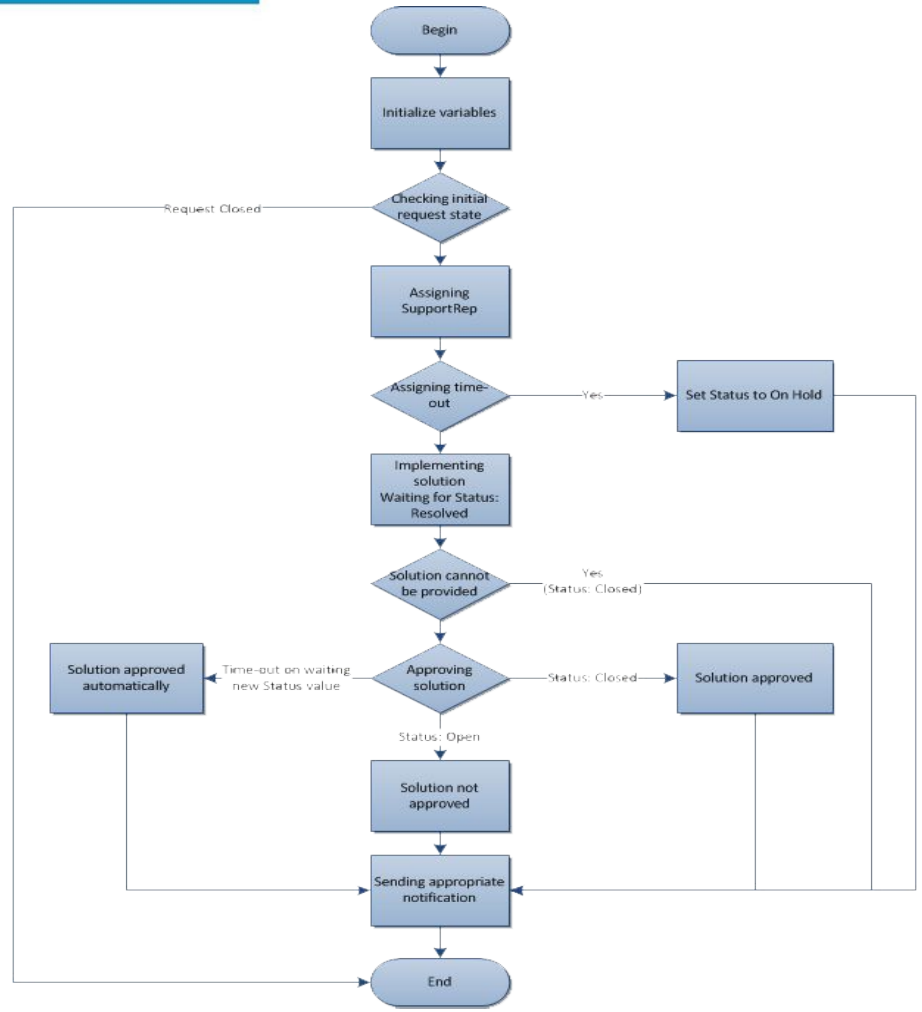
Недостатки словесного способа:

- строго не формализуем для машинного исполнения;
- избыточность;
- допускают неоднозначность толкования отдельных предписаний.

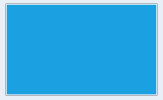
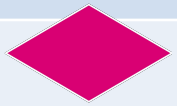
Графическая форма записи алгоритма

- Является более компактной и наглядной по сравнению со словесным;
- Изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий;

Такое графическое представление называется **схемой алгоритма** или **блок-схемой**.



Графическая форма записи алгоритма

Символ	Название	Назначение
	Данные	Обозначение процедуры ввода/вывода данных
	Процесс	Обработка данных в виде операции или группы операций (вычислительное действие или последовательность действий)
	Соединитель	Соединение прерванных линий потока
	Преопределенный процесс	Вычисление внутри выделенной подпрограммы/функции/модуле
	Подготовка	Изменение параметров цикла (организация счетчика)
	Решение	Проверка условия и организация ветвления потока
	Терминатор	Вход или выход во внешнюю среду (начало, конец алгоритма)
	Комментарий	Запись пояснений по алгоритму

Псевдокод

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. В нем не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования.

Примеры обозначений

алг (алгоритм)	сим (символьный)	дано	для	да
арг (аргумент)	лит (литерный)	надо	от	нет
рез (результат)	лог (логический)	если	до	при
нач (начало)	таб (таблица)	то	знач	выбор
кон (конец)	нц (начало цикла)	иначе	и	ввод
цел (целый)	кц (конец цикла)	все	или	вывод
вещ (вещественный)	длин (длина)	пока	не	утв

Псевдокод

Примеры записи алгоритмов:

алг Сумма квадратов (**арг цел** n, **рез це**
S)

дано | n > 0

надо | S = 1*1 + 2*2 + 3*3 + ... + n*n

нач цел i

ввод n; S = 0

нц для i **от** 1 **до** n

S = S + i*i

кц

вывод "S = ", S

кон

Алгоритм Дейкстры для нахождения кратчайших путей.

DIJKSTRA(G, l, s)

▷ Вход: граф $G(V, E)$ (ориентированный или нет) с неотрицательными
▷ длинами рёбер $\{l_e : e \in E\}$; вершина $s \in V$.
▷ Выход: для всех вершин u, достижимых из s,
▷ $dist[u]$ будет равно расстоянию от s до u (и ∞ для недостижимых).

for всех вершин $u \in V$
$dist[u] \leftarrow \infty$
$prev[u] \leftarrow$ NIL
$dist[s] \leftarrow 0$

$H \leftarrow$ MAKE-QUEUE(V) (в качестве ключей используются значения $dist$)
while $H \neq \emptyset$
$u \leftarrow$ DELETE-MIN(H)
for всех рёбер $(u, v) \in E$
if $dist[v] > dist[u] + l(u, v)$
then $dist[v] \leftarrow dist[u] + l(u, v)$
$prev[v] \leftarrow u$
DECREASE-KEY$(H, v, dist[v])$

Программная форма записи

Примеры программной записи алгоритмов

JavaScript

```
function sortBubble(data) {
   var tmp;
   for (var i = data.length - 1; i > 0; i--) {
      var counter=0;
      for (var j = 0; j < i; j++) {
         if (data[j] > data[j+1]) {
            tmp = data[j]; data[j] = data[j+1];
            data[j+1] = tmp; counter++;
         }
      }
      if(counter==0){ break; }
   }
 return data;
};
```

C++

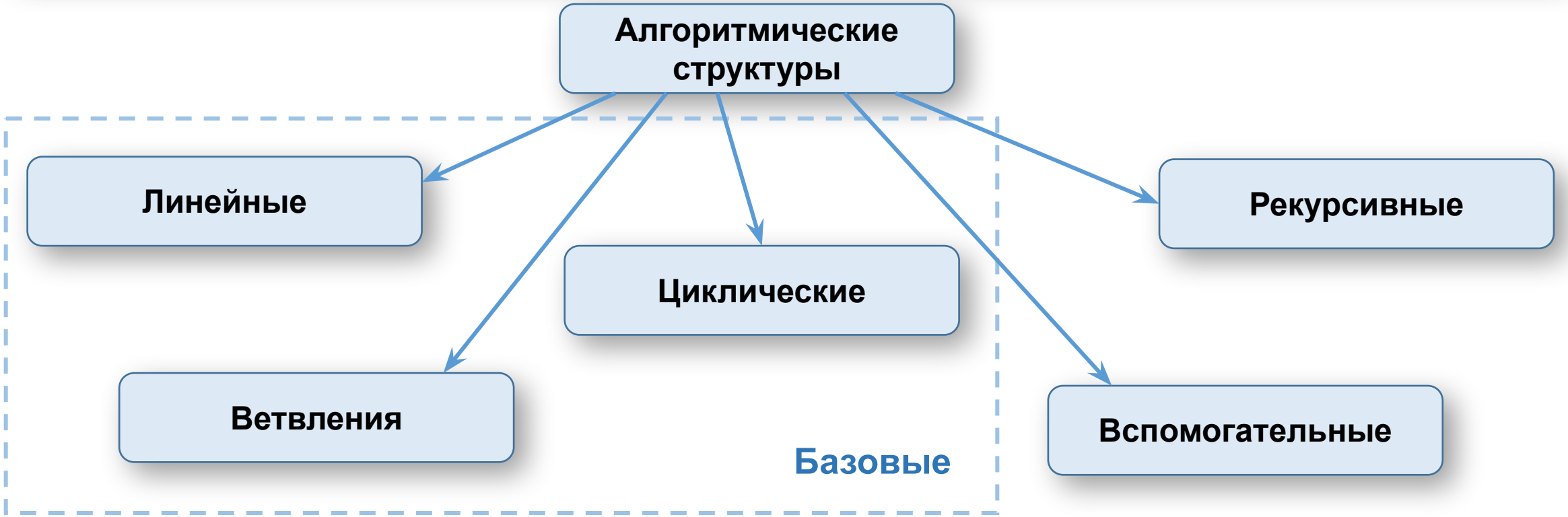
```
void bubble(int* a, int n) {
  for (int i = n - 1; i >= 0; i--) {
    for (int j = 0; j < i; j++) {
      if (a[j] > a[j + 1]) {
        int tmp = a[j];
        a[j] = a[j + 1];
        a[j + 1] = tmp;
      }
    }
  }
}
```

Правила составления схемы алгоритма

1. Каждый блок имеет один или несколько входов, кроме блока **"НАЧАЛО",** который не имеет входа вообще. Несколько стрелок, идущих на вход некоторого блока, положено соединять до точки входа в блок.
2. Каждый блок имеет строго один выход, кроме блока **"КОНЕЦ",** который не имеет выхода, и блока **проверки условия**, имеющего два выхода. Блок **проверки условия** обозначает разветвление вычислительного процесса в зависимости от выполнения некоторого условия. Чтобы различить выходы этого блока, их помечают словами **"да"** и **"нет".** Если условие, записанное в блоке, выполняется, то вычислительный процесс пойдет по стрелке со словом **"да",** иначе - по стрелке со словом **"нет".**

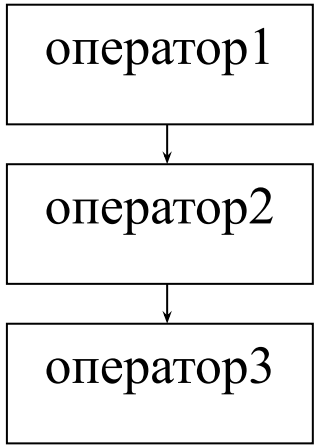
4. Базовые алгоритмические структуры

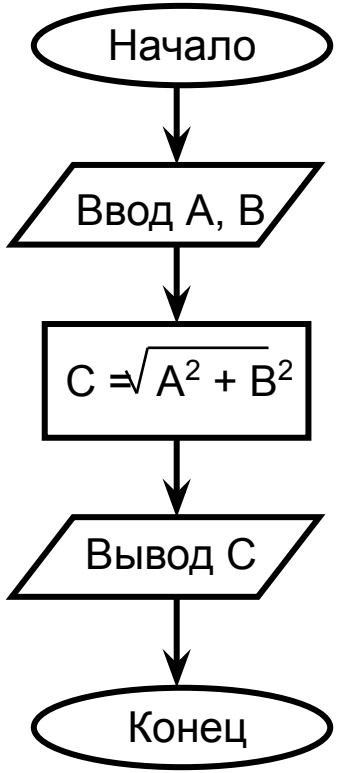
Алгоритмические структуры — типовые группы элементарных шагов алгоритма.



Линейная структура

Линейный алгоритм Р (или его часть) — если каждый шаг алгоритма выполняется только один раз, причем после каждого i-го шага выполняется (i + 1)-й шаг, если i-й шаг — не конец алгоритма.



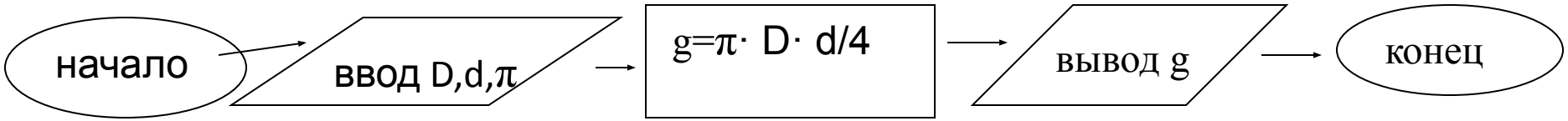


Алгоритмы линейной структуры

Пример 1.Требуется вычислить площадь поперечного сечения ствола по формуле эллипса g=π·D· d/4, где D - наибольший диаметр сечения ствола, d - наименьший диаметр сечения ствола, число π=3.1416.

Переменной называется величина, значение которой может меняться в процессе работы алгоритма. Каждой переменной отводится определённое место в памяти ЭВМ **(ячейка памяти).** В эту ячейку помещается текущее значение переменной. Вновь вычисленное значение переменной пересылается в ту же ячейку. При этом **"старое содержимое"** ячейки **теряется**.

Можно записать схему алгоритма так

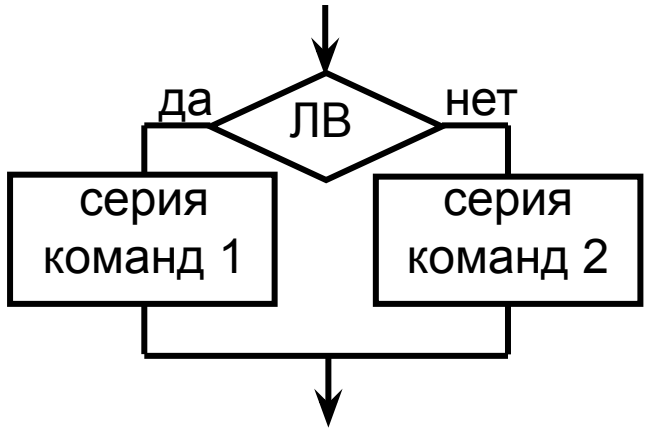


Ветвление

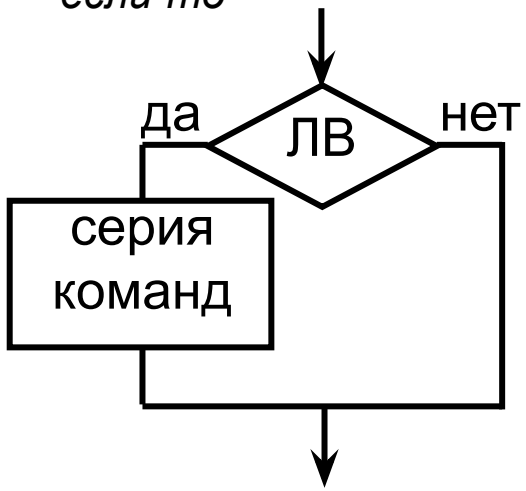
Ветвление — структура, где вычислительный процесс реализуется по одному из нескольких заранее предусмотренных направлений (**ветвей**) в зависимости от выполнения некоторого условия (логического выражения — ЛВ).

Ветвящийся процесс, включающий в себя две ветви, называется ***простым***, более двух ветвей - ***сложным***.

полное ветвление
если-то-иначе



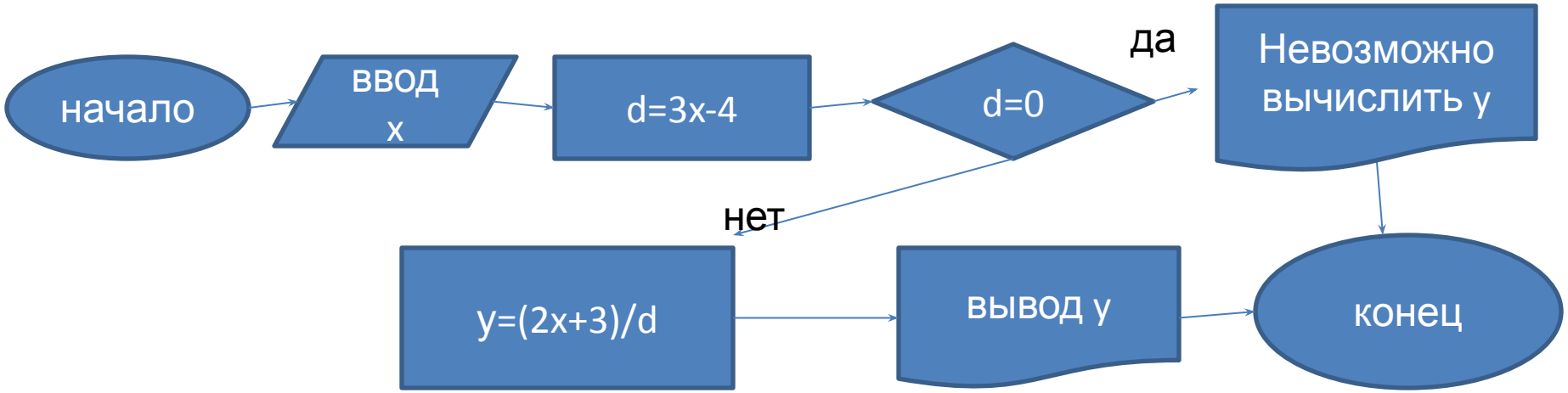
неполный вариант ветвления
если-то



Алгоритмы разветвляющейся структуры

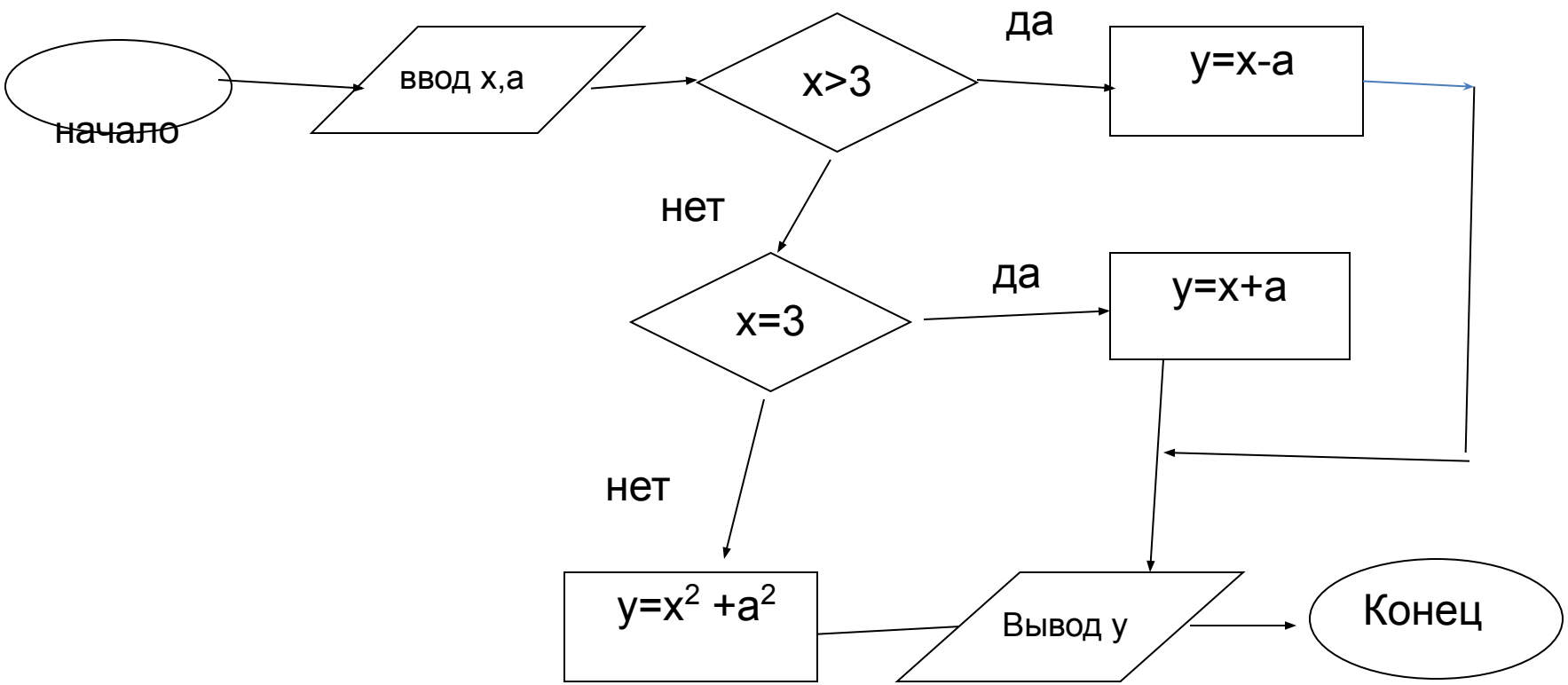
Пример 2. Составить схему алгоритма вычисления значения y=(2x+3)/(3x-4).

На первый взгляд, алгоритм нахождения значения y кажется линейным, но это не так. Приведём схему алгоритма.



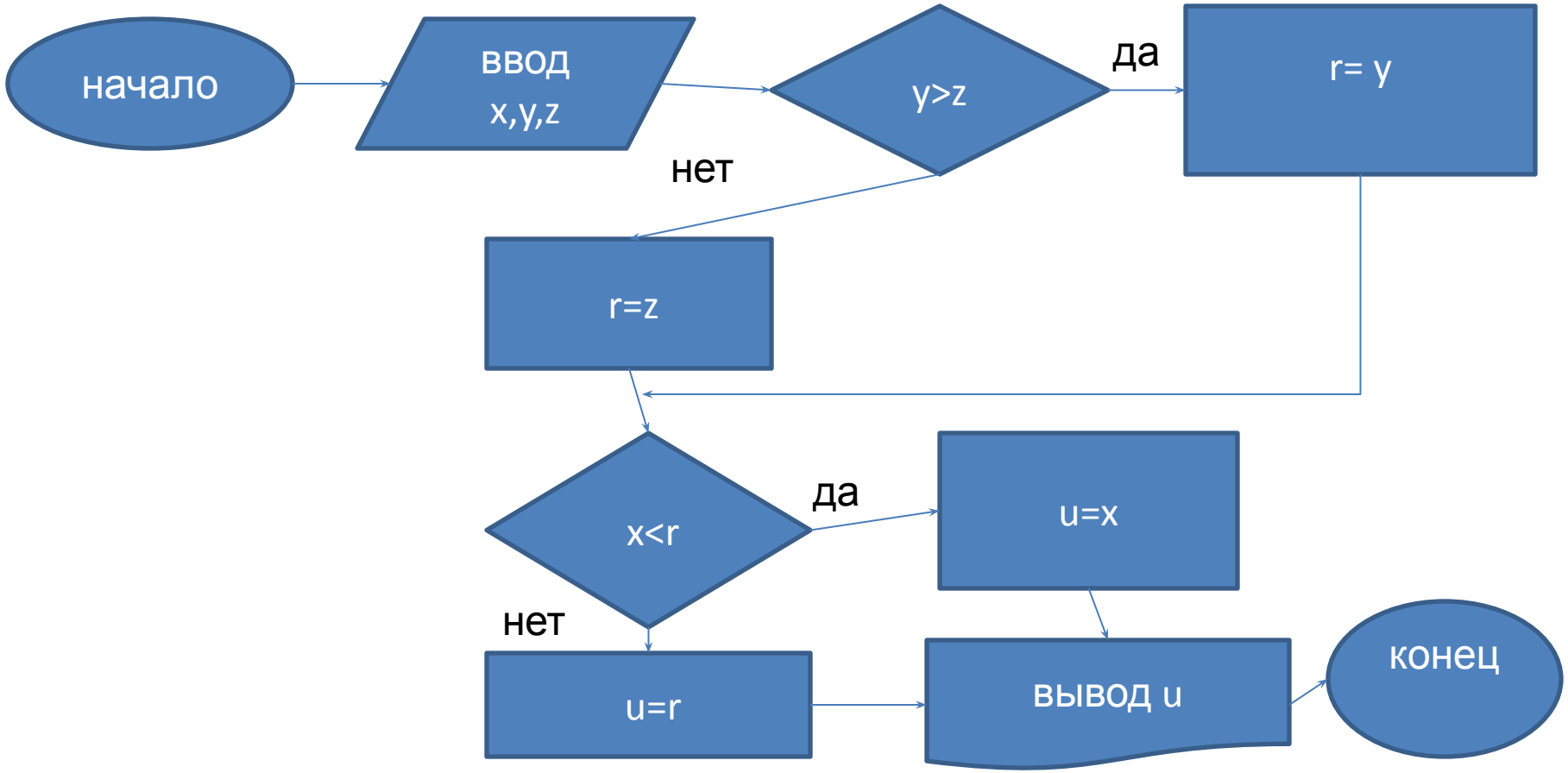
Алгоритмы разветвляющейся структуры

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



Алгоритмы разветвляющейся структуры

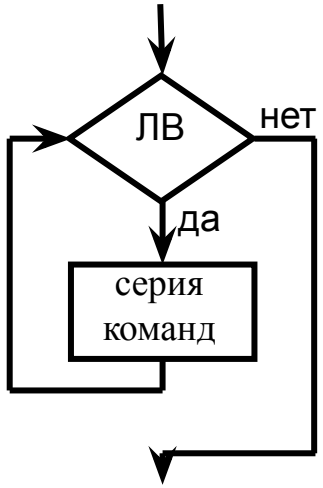
Пример 4. Даны различные x, y, z. Вычислить u=min(x,max(y,z))



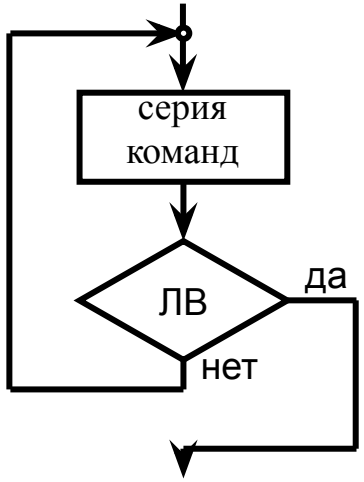
Циклические структуры

Цикл — структура, где подряд идущая группа шагов алгоритма, выполняется несколько раз. Количество повторений либо фиксировано, либо зависит от входных данных алгоритма.

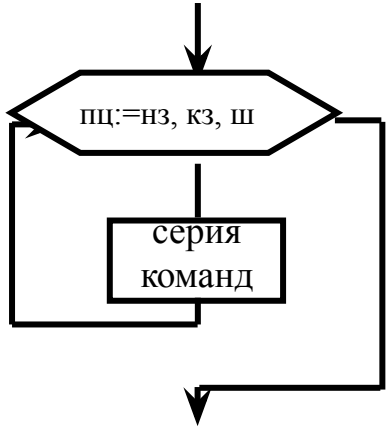
цикл с предусловием



цикл с постусловием

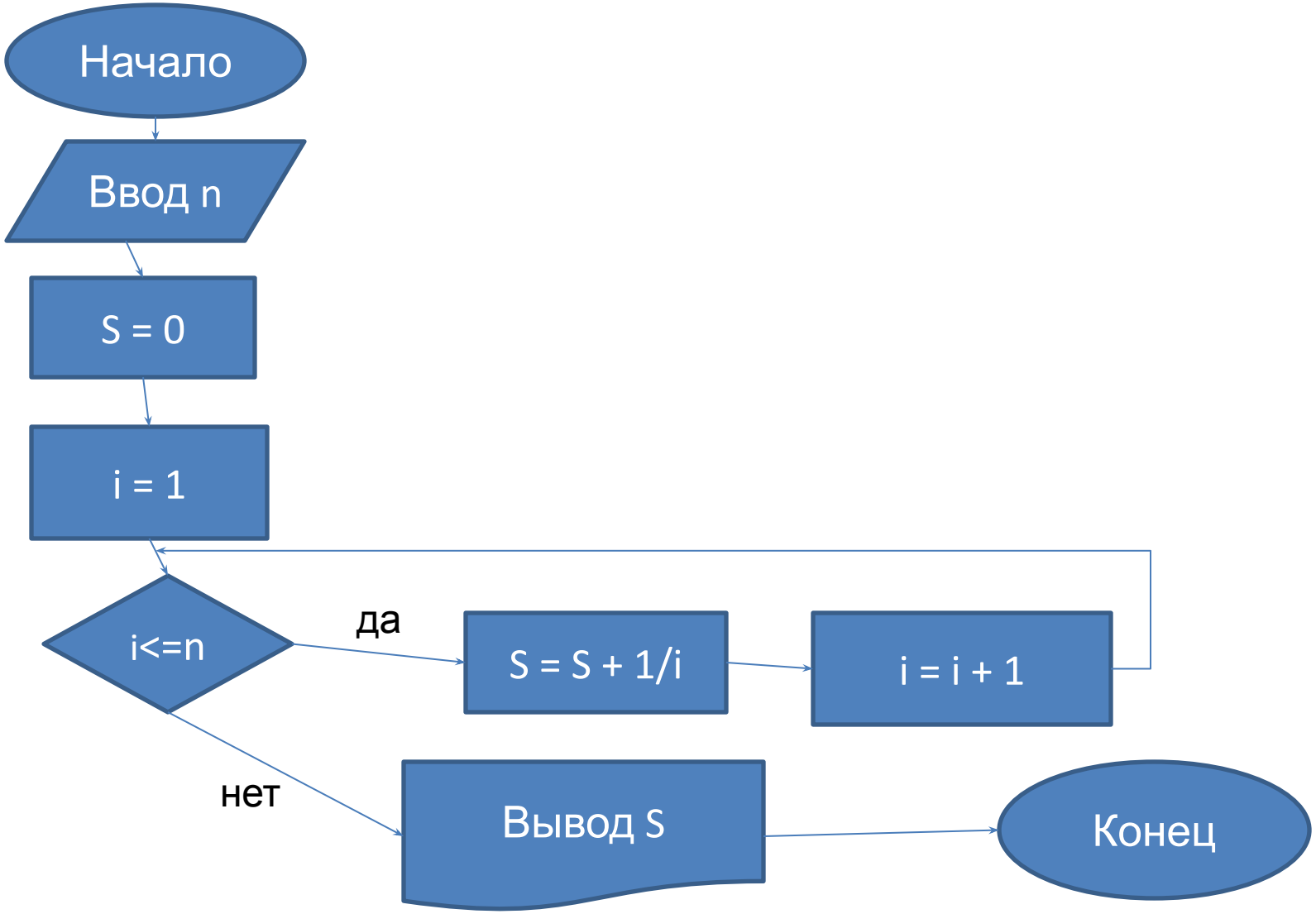


цикл со счетчиком



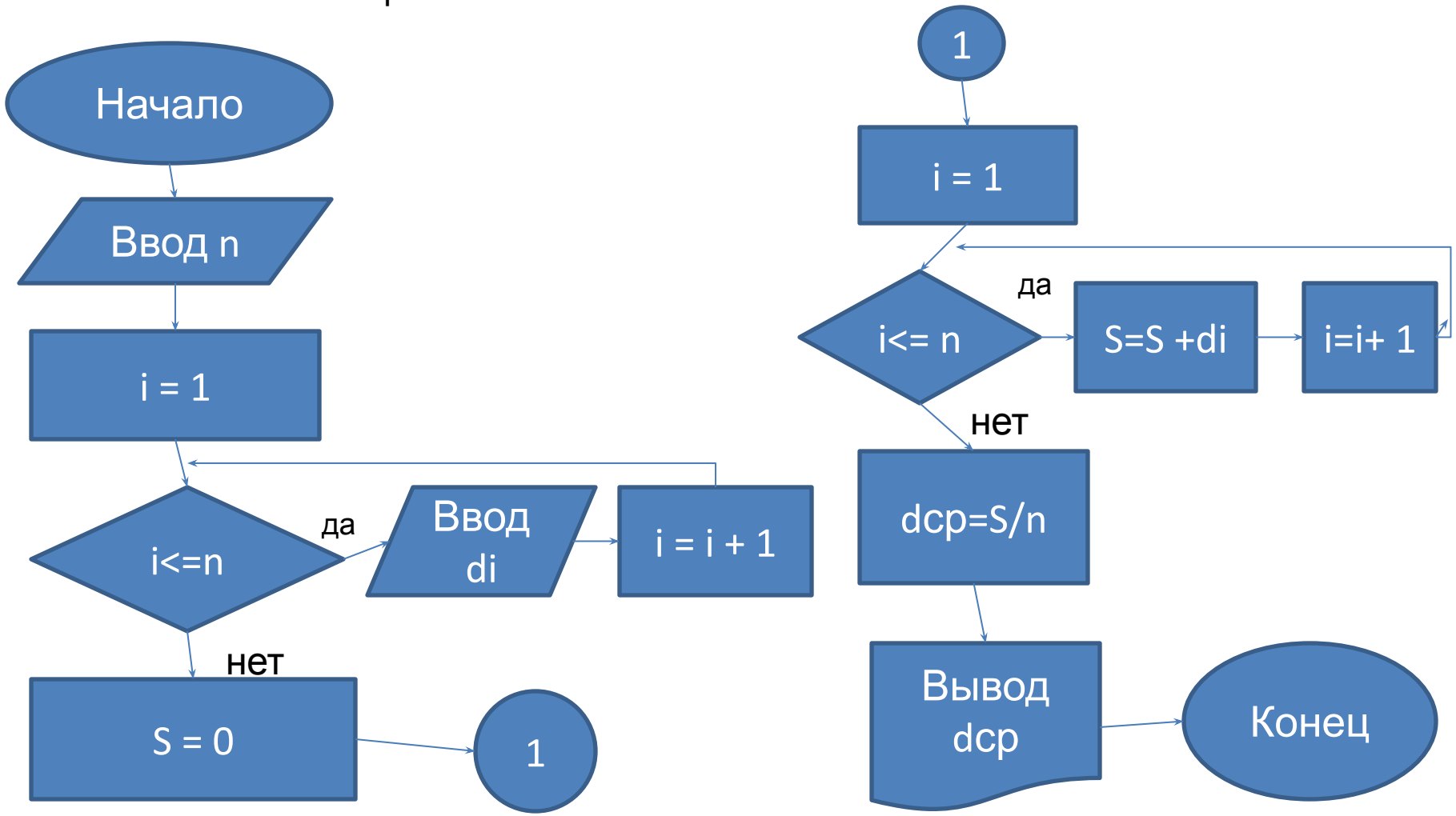
Алгоритмы циклической структуры

Пример 5. Найти конечную сумму S=1+1/2+1/3+….+1/n.



Алгоритмы циклической структуры

Пример 6. Дан массив чисел $D=(d_1,d_2,..,d_n)$. Найти $d_{ср}$ по формуле $d_{ср}= (d_1+d_2+..+d_n)/n$



Переменные и константы

Реальные данные, с которыми работает программа, - это:

- **числа;**
- **строки;**
- **логические величины** (аналоги 1 и О, "да" и "нет", "истина" и "ложь").

Эти типы данных называют базовыми.

Каждая **единица информации** хранится в ячейках памяти компьютера, имеющих свои **адреса**.

В языках программирования введено понятие **переменной**, позволяющее отвлечься от конкретных адресов и обращаться к содержимому памяти с помощью идентификатора или имени - последовательности, содержащей английские буквы, цифры, символы подчеркивания и начинающейся не с цифры.

6. Уровень языка программирования

Классификация языков программирования по критерию детализации предписаний:

- **машинные** (самого низкого уровня);
- машинно-ориентированные (**ассемблеры**);
- машинно-независимые (**высокого уровня**).

В зависимости от детализации предписаний определяют уровень языка программирования (чем меньше детализация, тем выше уровень). По данному критерию различают следующие языки программирования:
машинные (самого низкого уровня);
машинно-ориентированные (ассемблеры);
машинно-независимые (высокого уровня).

Машинные и машинно-ориентированные языки требуют подробного описания самых мелких деталей процесса обработки данных.

Язык ассемблера — это машинно-зависимый язык низкого уровня, в котором отдельным машинным командам соответствуют мнемонические (легко запоминаемые) имена, записываемые в текстовом виде.

Языки высокого уровня более разнообразны и интуитивно понятны для человека.
Преимущества:
универсальность - программа, написанная на таком языке, может выполняться на разных машинах;
независимость от аппаратуры, т.к. языки высокого уровня в значительной мере являются машинно-независимыми.