Лекция 2

Содержание

- Определение алгоритма.
- Базовые алгоритмические конструкции.
- Представление алгоритма: псевдокод, блок-схема.
- Итерация и рекурсия.

Алгоритмы (часть 1)

Определение алгоритма.

Алгоритм – это ***упорядоченный*** набор ***конечного*** числа строго ***определенных выполнимых*** шагов для решения ***задачи определенного типа*.**

Конечность.

Задача. Даны два отрезка разной длины *a* и *b*. Построить *c* - наибольший из отрезков, укладывающихся целое число раз в данных отрезках.
Алгоритм(?). Пусть $|a| > |b|$.
Шаг 1. Отложим отрезок *b* на отрезке *a* наибольшее количество раз.
Шаг 2. Если *b* точно отложился на *a* целое число раз, то выполнение алгоритма прекращается, задача решена, искомый отрезок – это *b*.
Шаг 3. Принять в качестве отрезка *b* остаток отрезка *a*, куда не помещался отрезок *b*, а в качестве отрезка *a* отрезок *b* и перейти к Шагу 1.

Является ли данный набор шагов алгоритмом?

Если существует некий отрезок, пусть очень малой длины, укладывающийся целое число раз в отрезках a и b, то можно среди таких отрезков найти и наибольший, используя приведенный выше набор шагов. Но может не существовать такого отрезка, в этом случае говорят, что отрезки несоизмеримы (отношение их длин выражается бесконечной непериодической десятичной дробью). Тогда последовательность приведенных шагов становится бесконечной. Таким образом, в общем случае вышеприведенный набор шагов не является алгоритмом. Для соизмеримых отрезков этот набор является алгоритмом (геометрический аналог алгоритма Евклида).

Вопросы:

Является ли метод деления столбиком алгоритмом нахождения частного? (бесконечные периодические дроби)

Является ли метод деления столбиком алгоритмом нахождения частного с заданной точностью?

То же для вавилонского метода оценки квадратного корня x из целого числа y:

x:=(x+y/x)/2.

Определенность.

Задача. Найти длину гипотенузы прямоугольного треугольника, зная длину его катетов, с точностью 0.01%.

Шаг 1. Возвести в квадрат длину 1-го катета

Шаг 2. Возвести в квадрат длину 2-го катета

Шаг 3. Сложить полученные числа.

Шаг 4. Найти число, квадрат которого с точностью 0.01% равен сумме квадратов длин катетов.

Является ли этот набор шагов алгоритмом?

Не определен шаг №4. Существует два числа, удовлетворяющие условию шага 4 и только одно из них положительное – арифметический квадратный корень. Необходимо детализировать этот шаг с тем, чтобы результат его выполнения был однозначным.

Выполнимость.

...

Шаг *N*. Умножить полученное число на сумму $x+y+z$, где (x,y,z) из N^3 является решением уравнения $x^4+y^4=z^4$ с наименьшим значением x.

...

Является ли этот набор шагов алгоритмом?

Во-первых, шаг *N* неоднозначен, при одном *x* может быть несколько решений с разными суммами. Во-вторых, шаг *N* содержит невыполнимые действия. Дело в том, что в соответствие с уже доказанной *теоремой Ферма* таких решений вообще не существует.

[*алгоритмически неразрешимые задачи; 10-я проблема Гильберта; вычислимость* [*А. Тьюринг*]]

Представление алгоритма.

Виды представлений.

Псевдокод, *pidgin Pascal, C* и т.п. **Блок-схема**. [диаграмма активности в UML] Программы на языке высокого уровня. **Программа** – последовательность нулей и единиц.

Базовые алгоритмические конструкции.

Присвоение.

<имя переменной>:=<выражение>

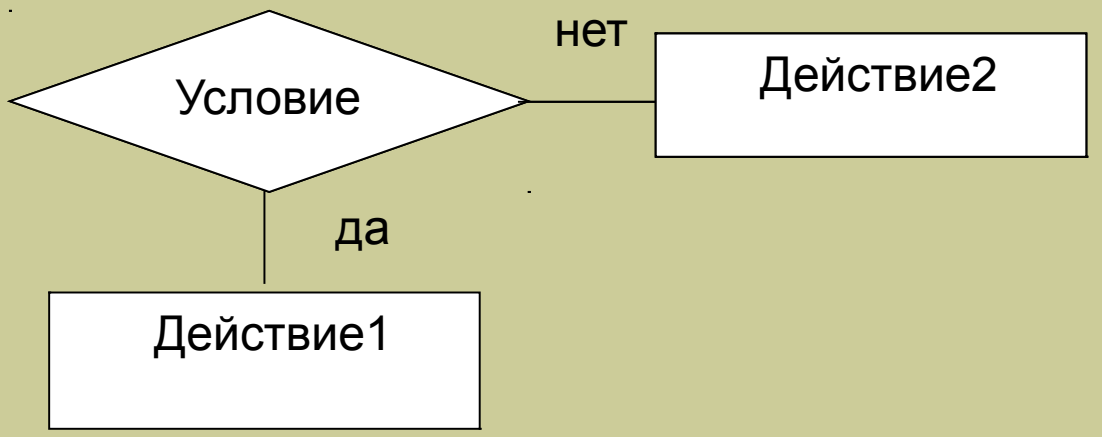
Следование.

<Действие 1>

<Действие 2>

Ветвление.

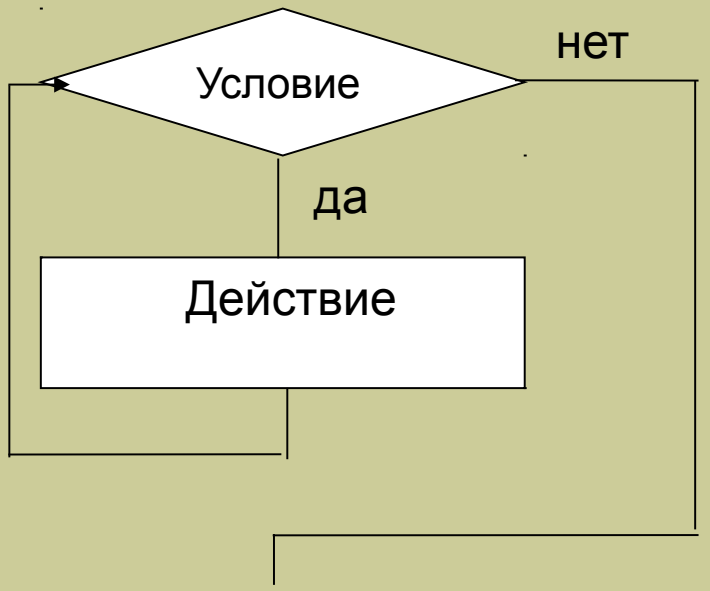
Если <условие>
то <действие 1>
иначе <действие 2>
Конец-если



Цикл-пока.

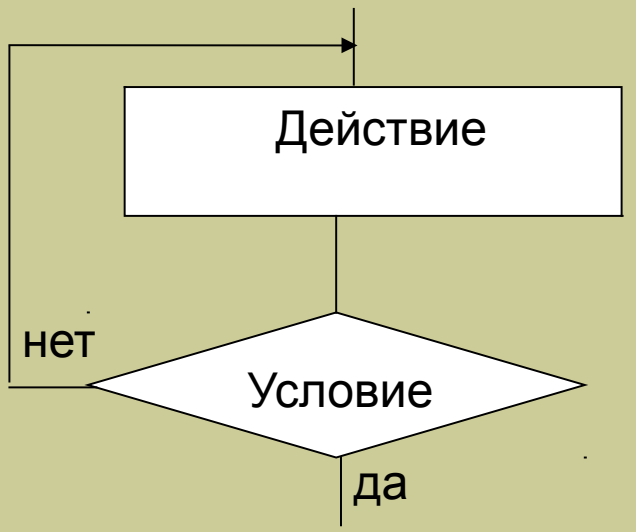
Цикл-пока <условие>

<действие>
Конец-цикл



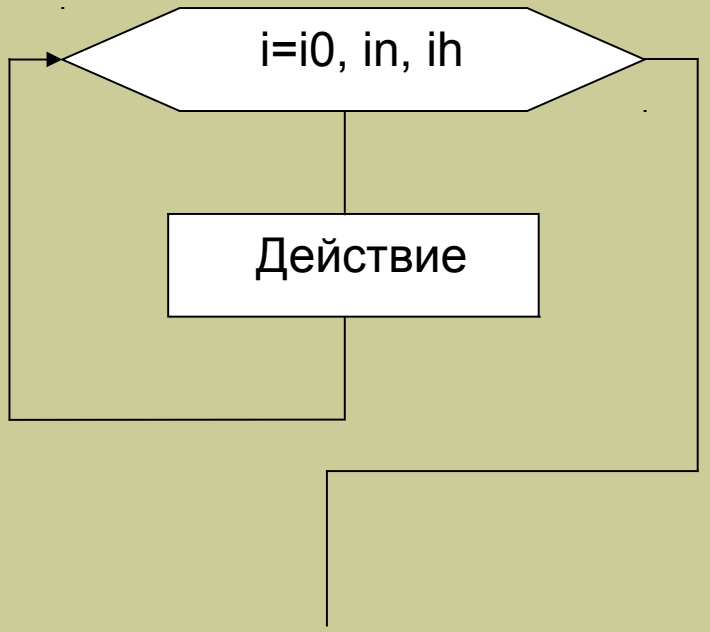
Цикл-до ***(****repeat****)***.

Выполнять
<Действие>
До <условие>



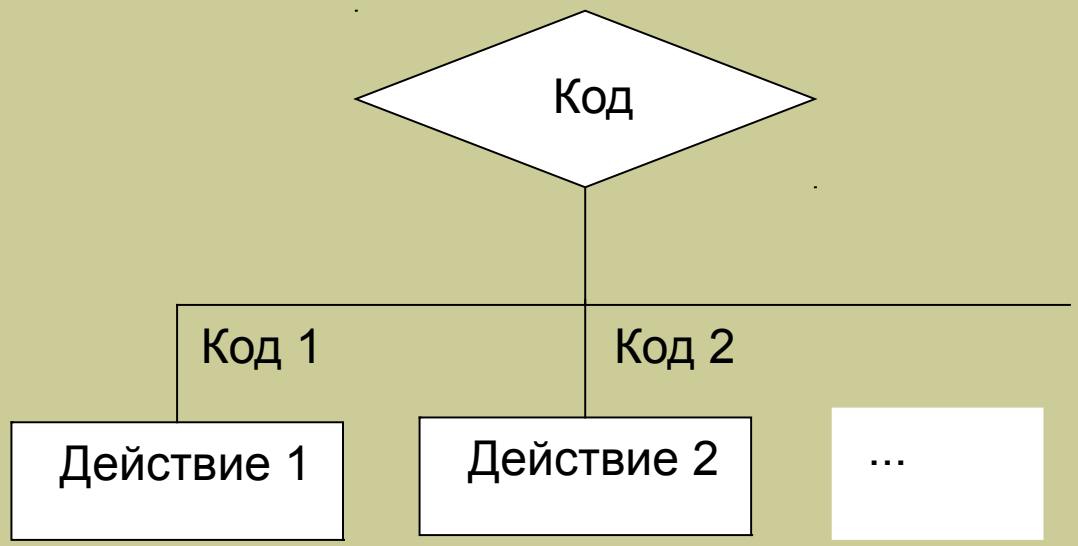
Счетный цикл

Для <индекс>=<i0, in, ih>
<Действие>
Конец-цикл



Выбор.

Выбор <код>
<код 1>:
<действие 1>
<код 2>:
<действие 2>
…
Конец-выбор



Процедура.

Описание процедуры:

Процедура <имя>(<список параметров>)

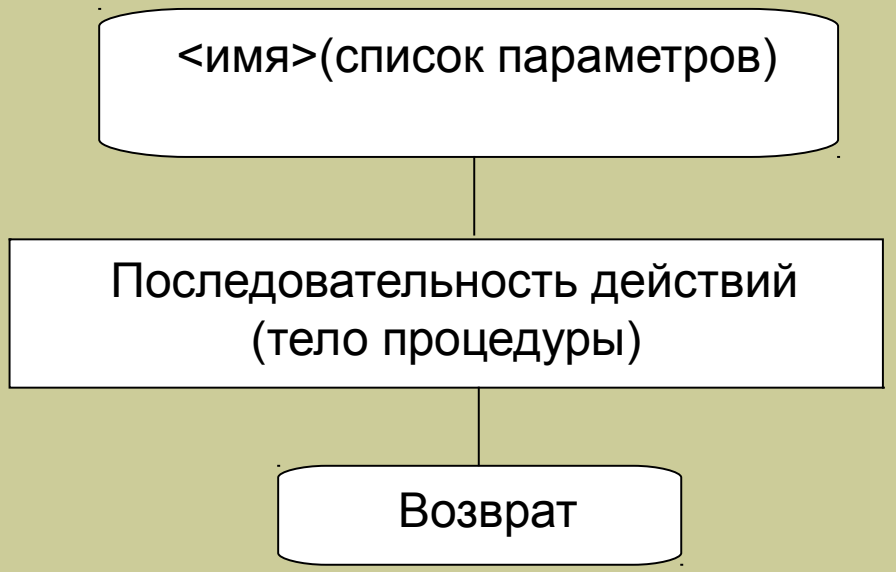
<действие 1>

<действие 2>

……..

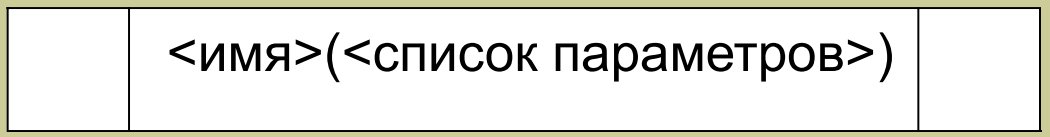
[<имя>:=<выражение>]

Возврат



Вызов процедуры:

<имя>(<список фактических параметров>)

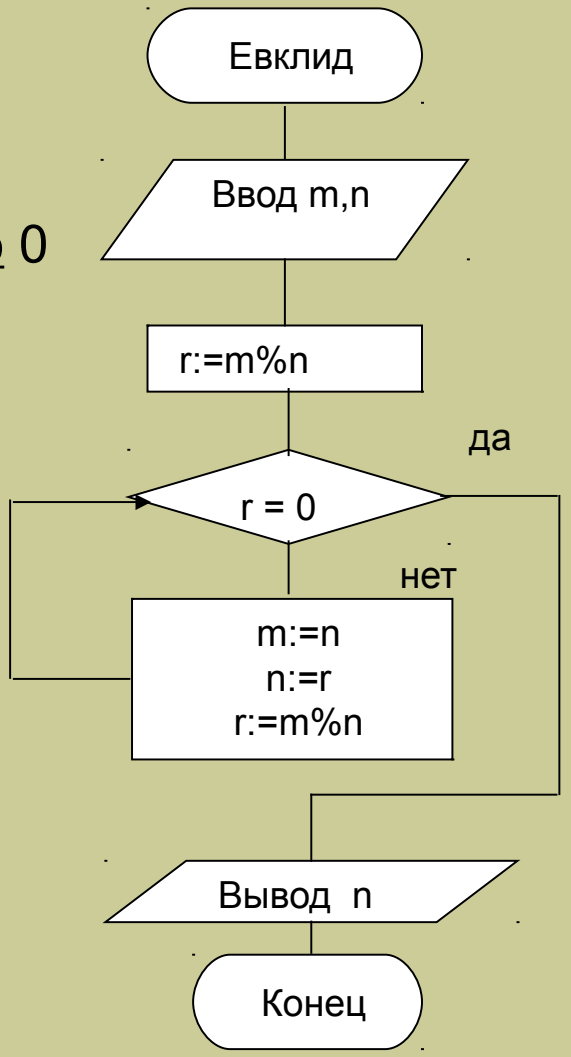


Упражнение

Представить алгоритм Евклида в виде псевдокода и блок-схемы.

Шаг1. Разделим m на n и пусть r – остаток.
Шаг2. Если r=0, то выполнение алгоритма прекращается; n – искомое значение.
Шаг3. Присвоить m:=n, n:=r и вернуться к шагу1.

Программа Евклид
  Ввод m, n
    r:=m%n
  Цикл-пока r не равно 0
    m:=n
    n:=r
    r:=m%n
  Конец-цикл
  Вывод n
Конец



Задание

- Будет ли алгоритм работать, если *m<n*?
- Всегда ли за конечное число шагов переменная *r* получит значение 0 ?
- Замените операцию деления по модулю - % (получение остатка) вычитанием.
- Оформите в виде процедуры совокупность действий, направленных на нахождение остатка.

Итерация и рекурсия

Итерация – от человека,
Рекурсия – от Бога.
-Л.Питер
Дойч

Определение процедуры:

Процедура <имя>(<список входных параметров; список выходных параметров>)

<действие 1>

<действие 2>

……

[<имя>:=<выражение>]

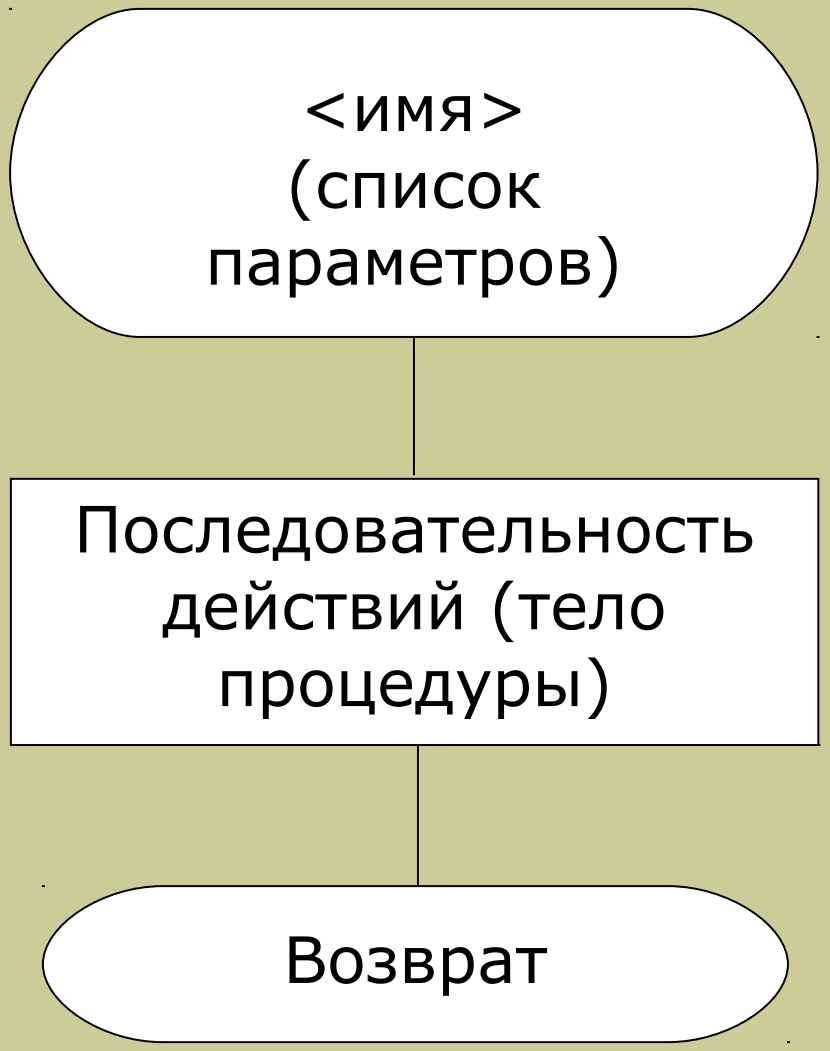
Возврат

Конец

Вызов процедуры:

<имя>(<список значений входных параметров>;
<список выходных параметров>)

Определение процедуры:

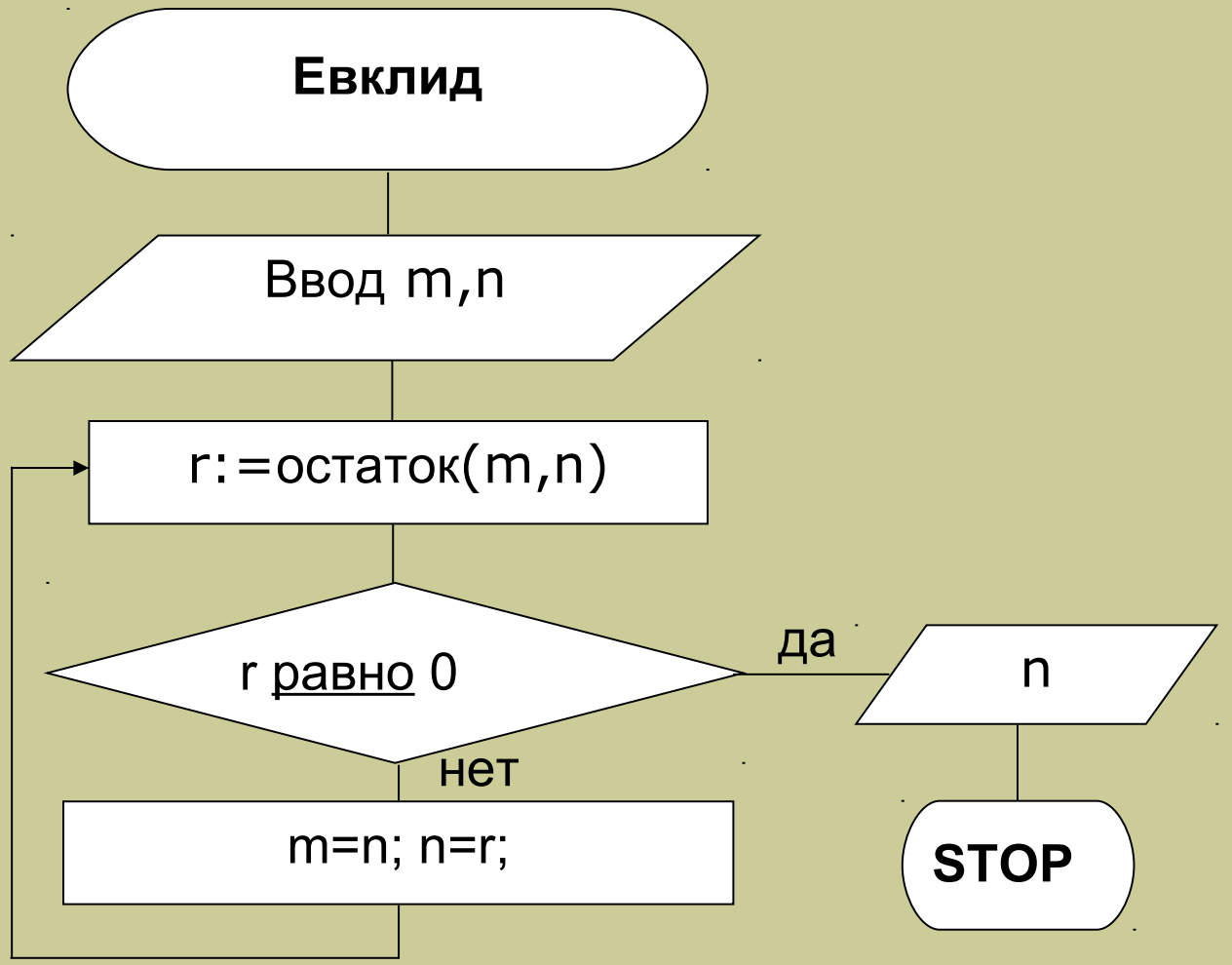


Вызов процедуры:

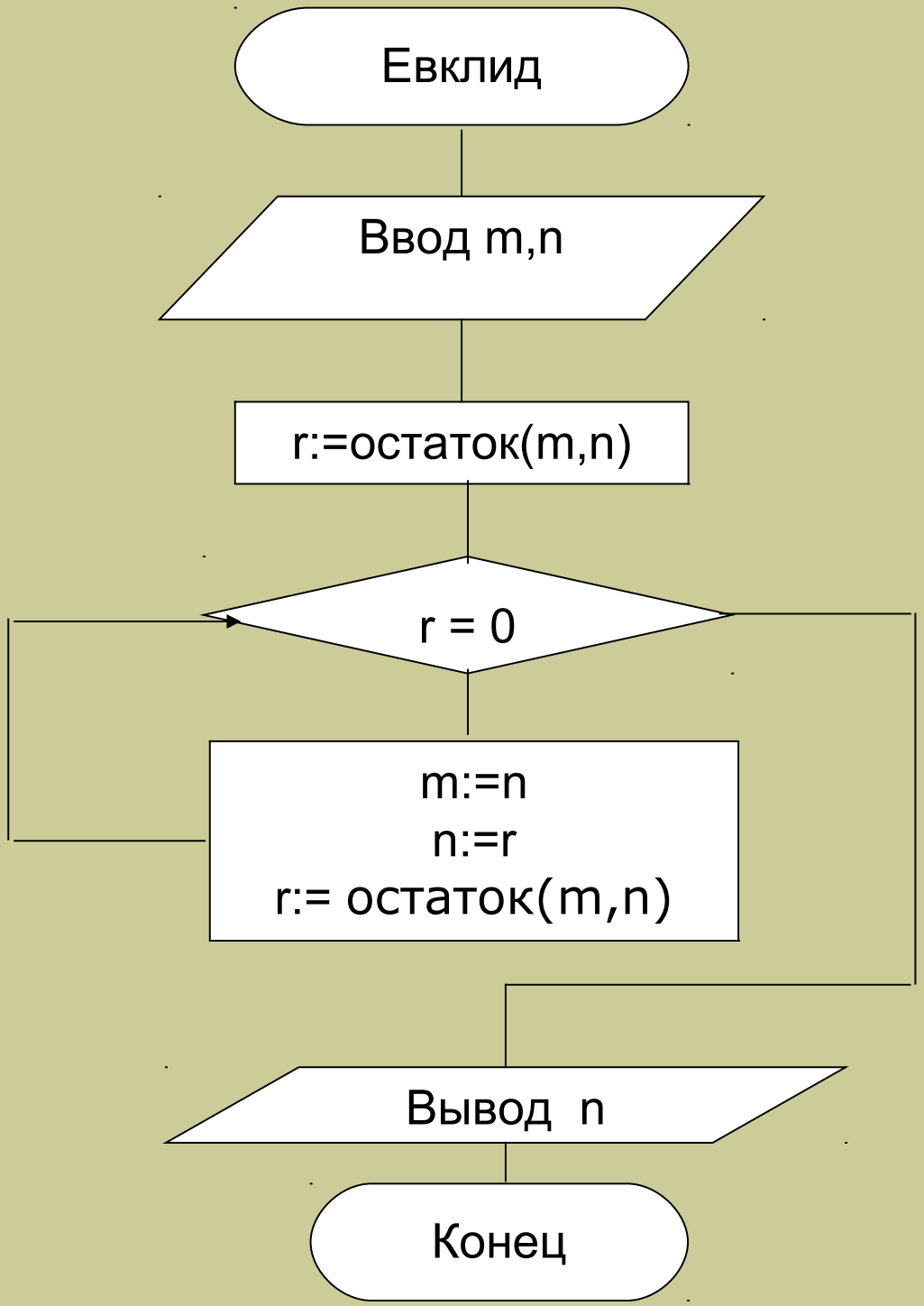
<имя>
(<список параметров>)

Введение процедуры позволяет создавать собственные стандартные блоки, которые, наряду с основными алгоритмическими конструкциями можно использовать при разработке алгоритмов.

```
Процедура  остаток(m, n)
        Цикл-пока  m >= n
           m:=m-n
        Конец цикла
        остаток:=m
        Возврат
Конец
```



Содержит ли блок-схема только основные алгоритмические конструкции?


Евклид
Ввод m,n
r:=остаток(m,n)
r = 0
m:=n
n:=r
r:= остаток(m,n)
Вывод n
Конец

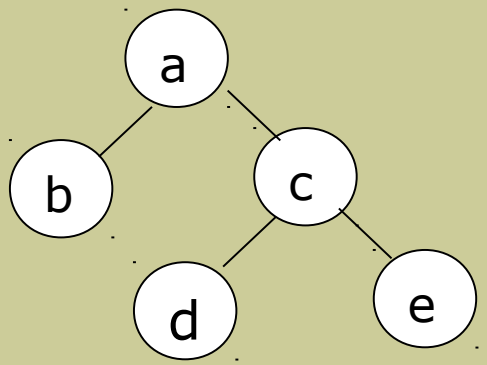
Введение понятия процедуры необходимо и достаточно для разработки ***рекурсивных алгоритмов****.*

Примеры рекурсивного определения:

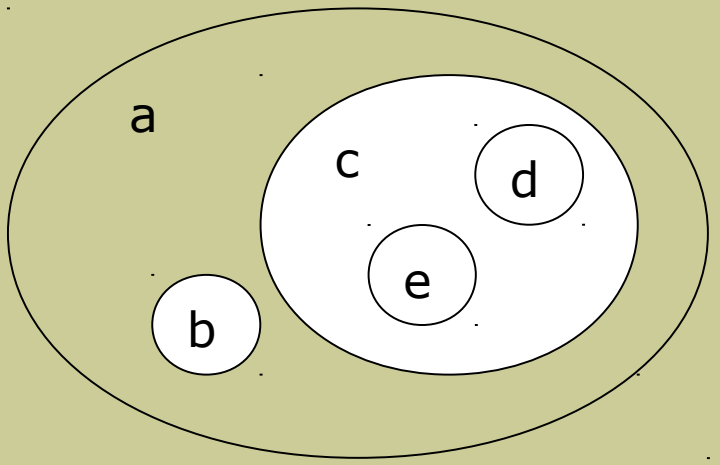
1. Матрёшка – это кукла, внутри которой находится матрёшка.
2. Если внутри куклы может находится больше одной матрешки, то такой объект структурно эквивалентен ***дереву***.

Древовидные структуры могут быть представлены различными способами –

- в виде диаграмм



или



- в виде вложенных скобок ***a(b,c(d,e))***
- в виде отступов

```
a
  b
  c
    d
    e
```

и т.д.

Пример рекурсивного определения функции (факториала):

$$F(n) = \begin{cases} 1 & , n = 0 \\ n * F(n-1) & , n > 0 \end{cases}$$

Алгоритм называется *рекурсивным*, если в его определении содержится прямое или косвенное использование этого же алгоритма.

Такие алгоритмы содержат процедуры с вызовом самой себя:

Прямая рекурсия:

```
Процедура рекурсия
……..
      рекурсия
……..
      Возврат
Конец
```

Косвенная рекурсия:

```
Процедура  рек1
……
      рек2
……
      Возврат
Конец
```

```
Процедура  рек2
……
      рек1
……
      Возврат
Конец
```

Если алгоритм содержит повторяющиеся действия, то их можно представить в *итеративной форме*, используя циклы, или в *рекурсивной форме*, используя рекурсивный вызов процедуры.

Рекурсивные процедуры, как и циклы, могут приводить к бесконечным вычислениям. Поэтому обращение к таким процедурам должно управляться некоторым условием, которое когда-то становится ложным.

```
Процедура  fact(n)
  Если n=0
  то
     fact:=1
  иначе
     fact:=n*fact(n-1)
  Конец-если
  Возврат
Конец
```

```
Процедура  fact(n)
  fact:=1
  Если n=0
  то  Возврат
  Конец-если
  Для    i:=1, n
     fact:=i*fact;
  Конец-цикл
  Возврат
Конец
```

Пусть имеется некий **Список** элементов. Надо построить алгоритм поиска хотя бы одного элемента с данным значением.

Последовательный алгоритм поиска:

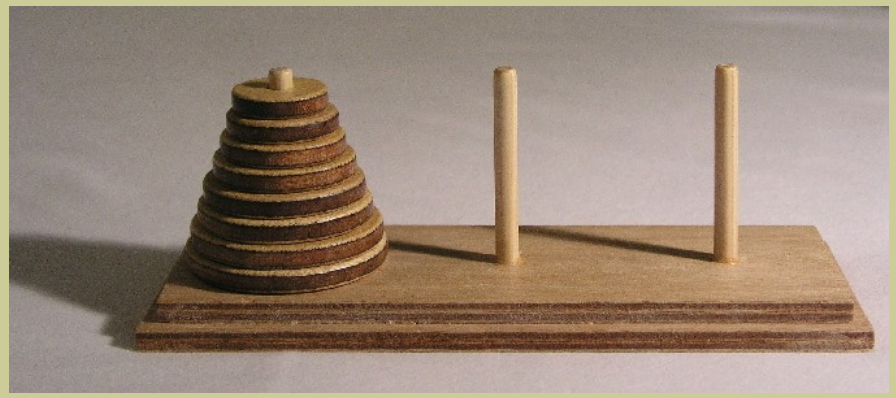
```
Процедура Поиск(список, искомое)
 Если список=пустой
 то    Вывод «Поиск неудачен»
 иначе
  проверяемое:= Первый_элемент_списка(список)
  Цикл-пока проверяемое не равно искомое и список не равно пустой
  проверяемое:=Следующее_значение_в_списке(список, проверяемое)
  Конец-цикл
  Если  проверяемое=искомое
  то    Вывод «Элемент имеется в списке»
  иначе  Вывод «Элемент отсутствует в списке»
  Конец-если
 Конец-если
 Возврат
Конец
```

Если **Список** упорядоченный по какому-то правилу, то для поиска элемента с данным значением можно построить более эффективный алгоритм.

Алгоритм двоичного поиска:

```
Процедура Поиск(список, искомое)
  Если список=пустой
  то    Вывод «Поиск неудачен»
  иначе
   проверяемое:= Средний_элемент_списка(список)
   Если  проверяемое= искомое
   то Вывод ««Элемент имеется в списке»
   иначе
       Если   проверяемое > искомое
       то
         Поиск(верхняя_половина_списка, искомое)
       иначе
         Если проверяемое < искомое
            Поиск(нижняя_половина_списка, искомое)
          Конец-если
        Конец-если
      Конец-если
  Конец-если
  Возврат
Конец
```

Благодарю за внимание



Задача о ханойских башнях

```
Процедура Ханой(n, a,b,c)
Если n=1
то
  a переместить на b
иначе
  Ханой(n-1, a,c,b)
  a переместить на b
  Ханой(n-1, c,b,a)
Возврат
Конец
```

```
Процедура Ханой(n, a,b,c;list)
 Если n=1
 то
   Добавить( (a,b), list)
 иначе
   Ханой(n-1, a,c,b; list1)
   Добавить(list1, list)
   Добавить( (a,b), list)
   Ханой(n-1, c,b,a; list2)
   Добавить(list2, list)
 Возврат
Конец
```

//list – список пар, задающих перемещения

Литература

1. Соболь Б.В. и др. Информатика. Ростов-на-Дону: Феникс, 2007, 447с.
2. Брукшир Дж. Информатика и вычислительная техника. М: ПИТЕР, 2004, 619с.
3. Вирт Н. Алгоритмы и структуры данных. М: Мир, 1989, 360с.
4. Кнут Д. Искусство программирования. Т1. Основные алгоритмы. М: Вильямс, 712с.