Тема 3.

Ветвления. Массивы. Циклы

© С.В.Кухта, 2009

Содержание

1. Простые и сложные условия
2. Операторы ветвления: ***if*** и ***case***.
3. Массивы: описание и использование.
4. Операторы цикла: ***for***, ***while*** и ***repeat***.
5. Прерывание циклов: ***break***, ***continue*** и ***goto***.
6. Решение некоторых типовых задач

© С.В.Кухта, 2009

1. Условия в языке Паскаль

© С.В.Кухта, 2009

Условия используются в программах для организации ветвлений и повторяющихся действий.

Условием является логическое выражение – выражение типа ***Boolean***: ***True*** (истина) и ***False*** (ложь).
Булевское значение дает любая из операций отношений:

= <> <= < > >= in

Условия классифицируются как простые и сложные.

© С.В.Кухта, 2009

Простые условия

Простые условия состоят из одного из следующих элементов:

- логического значения;
- логической функции;
- выражений, соединенных знаком отношения.

```
True     False        Flag
Odd(X)  Pred(True)     Odd(A*P + B)
A+B<>0
(K Mod 5) = 0    Number div Modulo = 2
Sin(2*x) > S     b*b > 4*a*c
(X + Y) mod Prime = 0
```

© С.В.Кухта, 2009

Сложные условия

Сложные условия конструируются из простых с помощью логических операций:

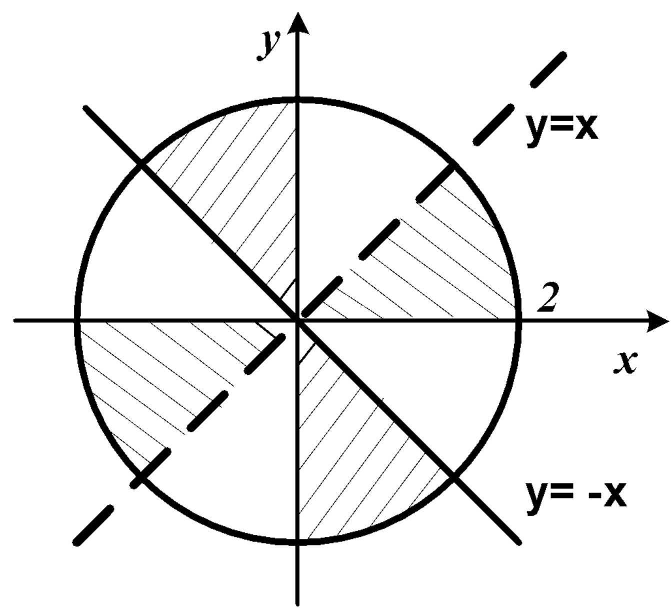
- ***Not*** – логическое отрицание (НЕ);
- ***And*** – логическая конъюнкция (И);
- ***Or*** – логическая дизнъюнкция (ИЛИ);
- ***Xor*** – логическое исключающее ИЛИ.

Таблицы истинности (значений операций) приведены в Теме 2.

```
A + i > B) or ( X [Index] = C )
{Здесь A, B, C – переменные типа Real,
   X – массив вещественных чисел,
   Index – переменная типа Integer }
Not(beta) And (gamma)
{beta и gamma – переменные типа Boolean}
(A > B) = (C > D)
{результат операций A>B и C>D – переменные типа
Boolean}
```

Пример построения условий

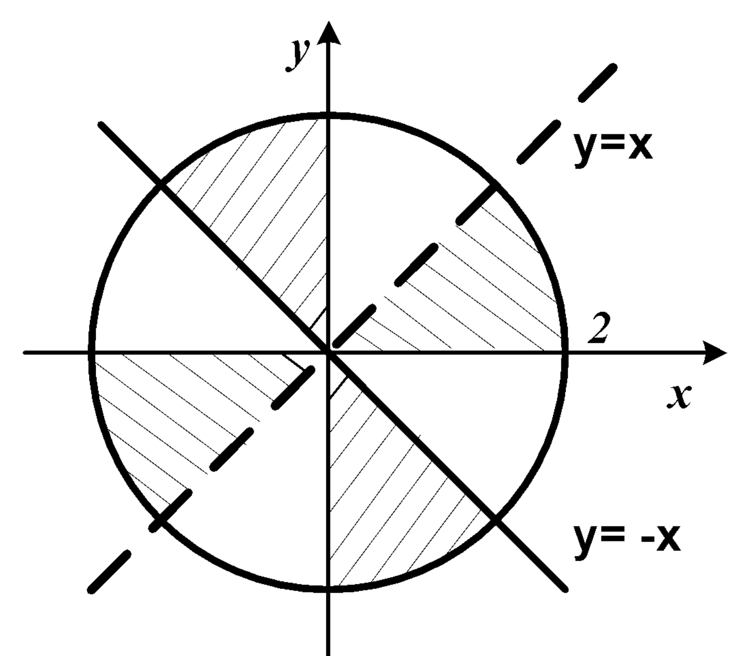
Запишите логическое выражение, которое принимает значение "истина" тогда и только тогда, когда точка с координатами (x, y) принадлежит заштрихованной области.



точки на границе не входят в область

точки на границе входят в область

Пример построения условий



Решение:

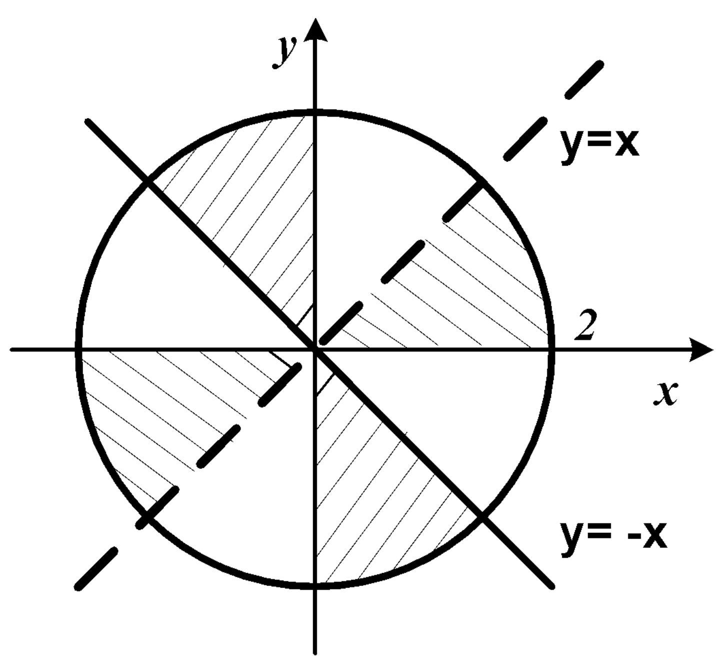
Заштрихованная область образуется графиками прямых $y=x$, $y=-x$ и окружности $x^2 + y^2 = 2^2$.

Для всех заштрихованных секторов справедливо $x^2 + y^2 \le 2^2$.

Для 1-й четверти: $y < x$ и $x \ge 0$ и $y \ge 0$.
Для 2-й четверти: $y \ge -x$ и $x \le 0$ и $y \ge 0$.
Для 3-й четверти: $y > x$ и $x \le 0$ и $y \le 0$.
Для 4-й четверти: $y \le -x$ и $x \ge 0$ и $y \le 0$.

© С.В.Кухта, 2009

Пример построения условий



Ответ:

*(x*x + y*y ≤ 4)* ***and*** *((y < x)* and *(x >= 0)* and *(y >= 0)*
or (y >= -x) and *(x <= 0)* and *(y >= 0)*
or (y > x) and *(x <= 0)* and *(y <= 0)*
or (y <= -x) and *(x >= 0)* and *(y <= 0))*

© С.В.Кухта, 2009

2. Операторы ветвления

© С.В.Кухта, 2009

К операторам, позволяющим из нескольких возможных вариантов выполнения программы (ветвей) выбрать только один, относятся ***if*** и ***case***.

Т.е. эти операторы позволяют изменить естественный порядок выполнения операторов программы.

© С.В.Кухта, 2009

Полный условный оператор *if*

Формат:

```
if   <условие>   then   < оператор 1 >
   else <оператор 2> ;
```

Схема выполнения оператора:

А если "ложь" (***false***) - без дополнительных проверок выполняется оператор, стоящий после ключевого слова ***else***

Затем, если в результате получена "истина" (***true***), то выполняется оператор, стоящий после ключевого слова ***then***

Обратите внимание, что перед словом ***else*** символ **";"** не ставится - ведь это разорвало бы оператор на две части.

© С.В.Кухта, 2009

Полный условный оператор *if*

Примеры:

```
if  a>=b  then  Max:=a  else  Max:=b;

 If  (x<= 0) And (y>5)  then  begin
      u:=x*x-2*y+3;  v:=1/2*x+1
   end
 else begin
   u:=1/3*x+2;  v:=x*x+3*y-2
 end;

If (x*x+y*y ≤4) and ((y<x) and (x>=0) and (y>=0)
   or (y>=-x) and (x<=0) and (y>=0)
   or (y>x) and (x<=0) and (y<=0)
   or (y<=-x) and (x>=0) and (y<=0))  then
      writeln ('точка принадлежит области')
else  writeln ('точка не принадлежит области');
```

© С.В.Кухта, 2009

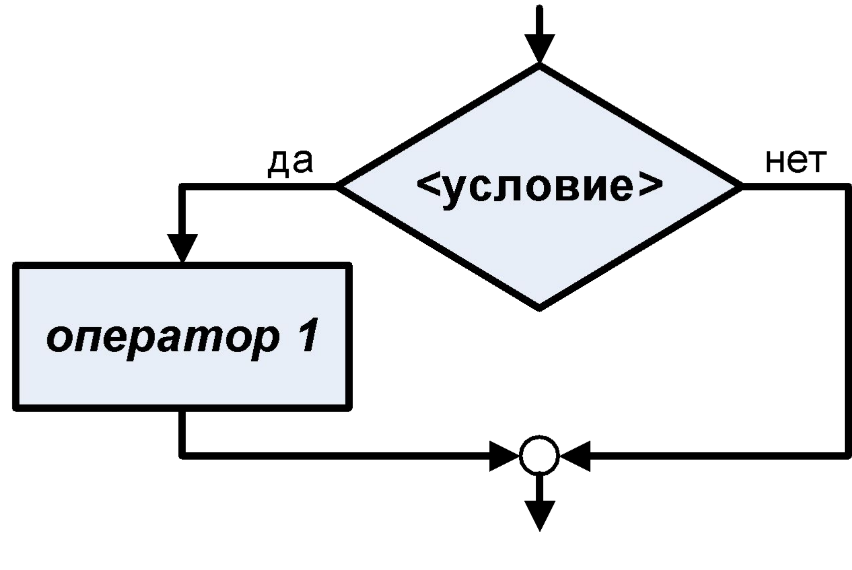
Неполный условный оператор *if*

Формат:

```
if   <условие>   then   < оператор 1 >;
```

Если условие истинно, то выполняется *<оператор 1>*, если ложно – оператор, следующий сразу за оператором ***if***.

Схема выполнения оператора:



© С.В.Кухта, 2009

Неполный условный оператор *if*

Примеры:

```
If  F mod 3 = 0 then write(i);

 If  (x<= 0) And (y>5)  then  begin
   u:=x*x–2*y+3;  v:=1/2*x+1  end;

If (x*x+y*y ≤4) and ((y<x) and (x>=0) and (y>=0)
   or (y>=-x) and (x<=0) and (y>=0)
   or (y>x) and (x<=0) and (y<=0)
   or (y<=-x) and (x>=0) and (y<=0))  then
      writeln ('точка принадлежит области');
```

© С.В.Кухта, 2009

В операторе ***if*** по обеим ветвям (***then*** и ***else***) может выполняться только один оператор!

При необходимости выполнения нескольких требуется использовать операторные скобки ***begin-end***.

© С.В.Кухта, 2009

Многозначные ветвления

Что же произойдет, если написать несколько вложенных операторов *if*?

В случае, когда каждый оператор ***if*** имеет собственную ***else***-ветвь, все будет в порядке.

А вот если некоторые из них этой ветви не имеют, может возникнуть ошибка.

© С.В.Кухта, 2009

Многозначные ветвления

Компилятор языка Паскаль всегда считает, что ***else*** относится к самому ближнему оператору ***if***. Таким образом, если написать

```
if i>0 then if s>2
             then s:= 1
            else s:= -1;
```

подразумевая, что ***else***-ветвь относится к внешнему оператору ***if***, то компилятор все равно воспримет эту запись как

```
if i>0 then if s>2
            then s:= 1
   else s:= -1        else;
```

Ясно, что таким образом правильного результата получить не удастся.

© С.В.Кухта, 2009

Многозначные ветвления

Для того чтобы избежать подобных ошибок, стоит всегда (или по крайней мере при наличии нескольких вложенных условных операторов) указывать оба ключевых слова, даже если одна из ветвей будет пустовать.

Так вы застрахуетесь от одной из частых "ошибок по невнимательности", которые очень сложно найти в процессе отладки программы.

© С.В.Кухта, 2009

Многозначные ветвления

Итак, исходный вариант нужно переписать следующим образом:

```
if i>0 then if s>2
         then s:= 1
         else
        else s:= -1;
```

либо так:

```
if i>0 then begin if s>2
             then s:=1
   end
   else s:=-1;
```

Вообще же, если есть возможность переписать несколько вложенных условных операторов как один оператор ***выбора***, это стоит сделать.

© С.В.Кухта, 2009

Пример. Корни квадратного уравнения

```
Program SquareEquation;
 Var   a, b, c, Root1, Root2, Discriminant : real;
    Solution: Integer;
Begin
   Write('Введите коэффициенты   уравнения ');
   Readln(a, b, c) ;
   Discriminant := Sqr(b)-4*a*c;
   If  Discriminant<0  then { Нет корней } Solution:=0
   else   If  Discriminant=0 then begin  { Один корень }
            Solution := 1;  Root1 := -b/a;
            Writeln (' x1= ' ,Root1)   end
          else    { Два корня }    begin
         Solution := 2;
            Root1 :=(-b+Sqrt(Discriminant))/(2*a);
         Root2 :=(-b-Sqrt(Discriminant))/(2*a);
         Writeln(' x1= ', Root1, ' x2= ', Root2)
         end ;
   Writeln(' Количество решений равно: ', Solution)
End.
```

Разветвляющиеся алгоритмы

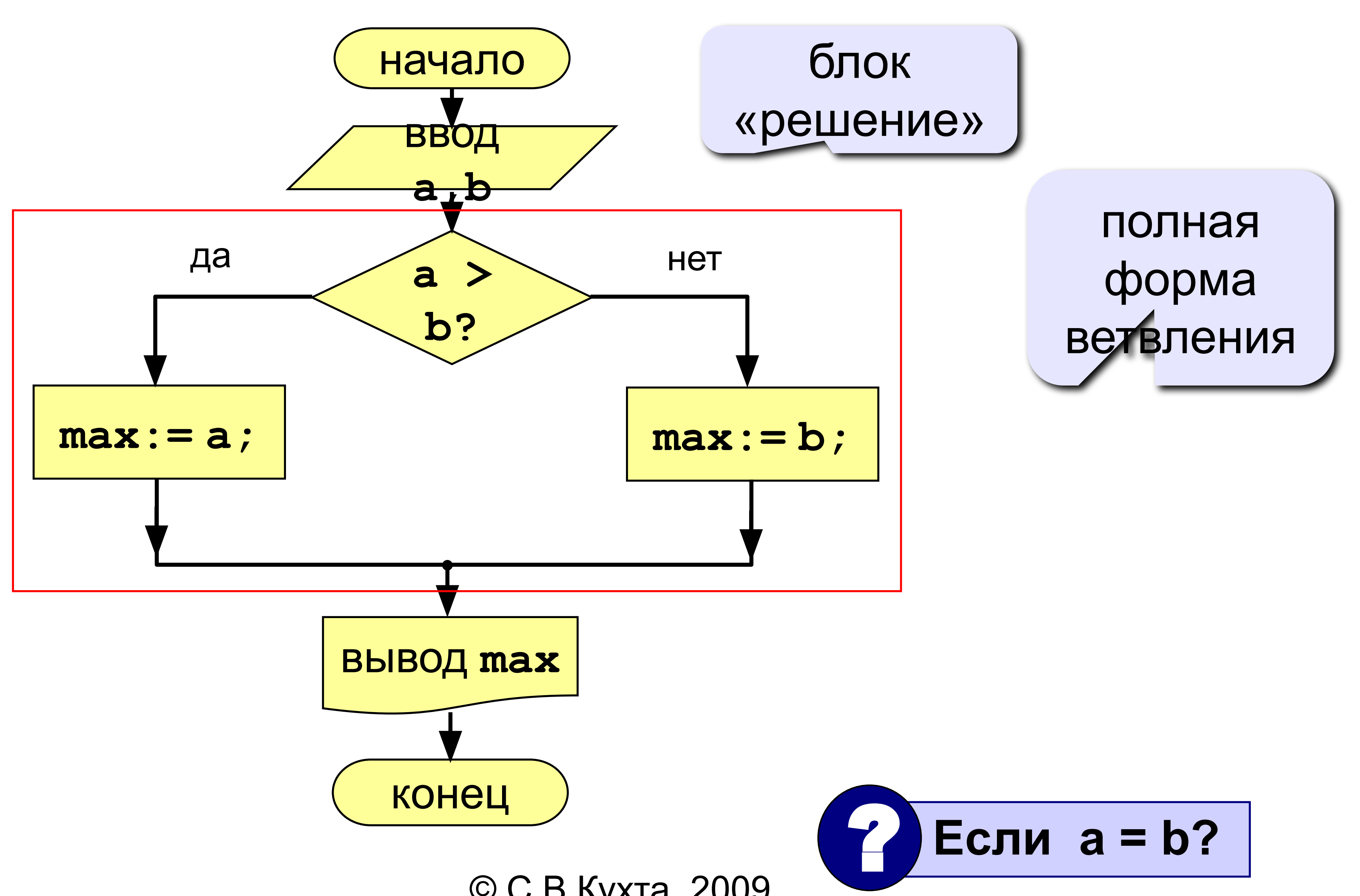
Задача. Ввести два целых числа и вывести на экран наибольшее из них.

Идея решения: надо вывести на экран первое число, если оно больше второго, или второе, если оно больше первого.

Особенность: действия исполнителя зависят от некоторых условий (***если … иначе …***).

© С.В.Кухта, 2009

Вариант 1. Блок-схема



© С.В.Кухта, 2009

Вариант 1. Программа

```
program qq;
var a, b, max: integer;
begin
  writeln('Введите два целых числа');
  read ( a, b );
  if a > b then begin
    max := a;
  end
  else begin
    max := b;
  end;
  writeln ('Наибольшее число ', max);
end.
```

полная форма условного оператора

© С.В.Кухта, 2009

Что неправильно?

```
if a > b then begin
  a := b;
end
else begin
  b := a;
end;
```

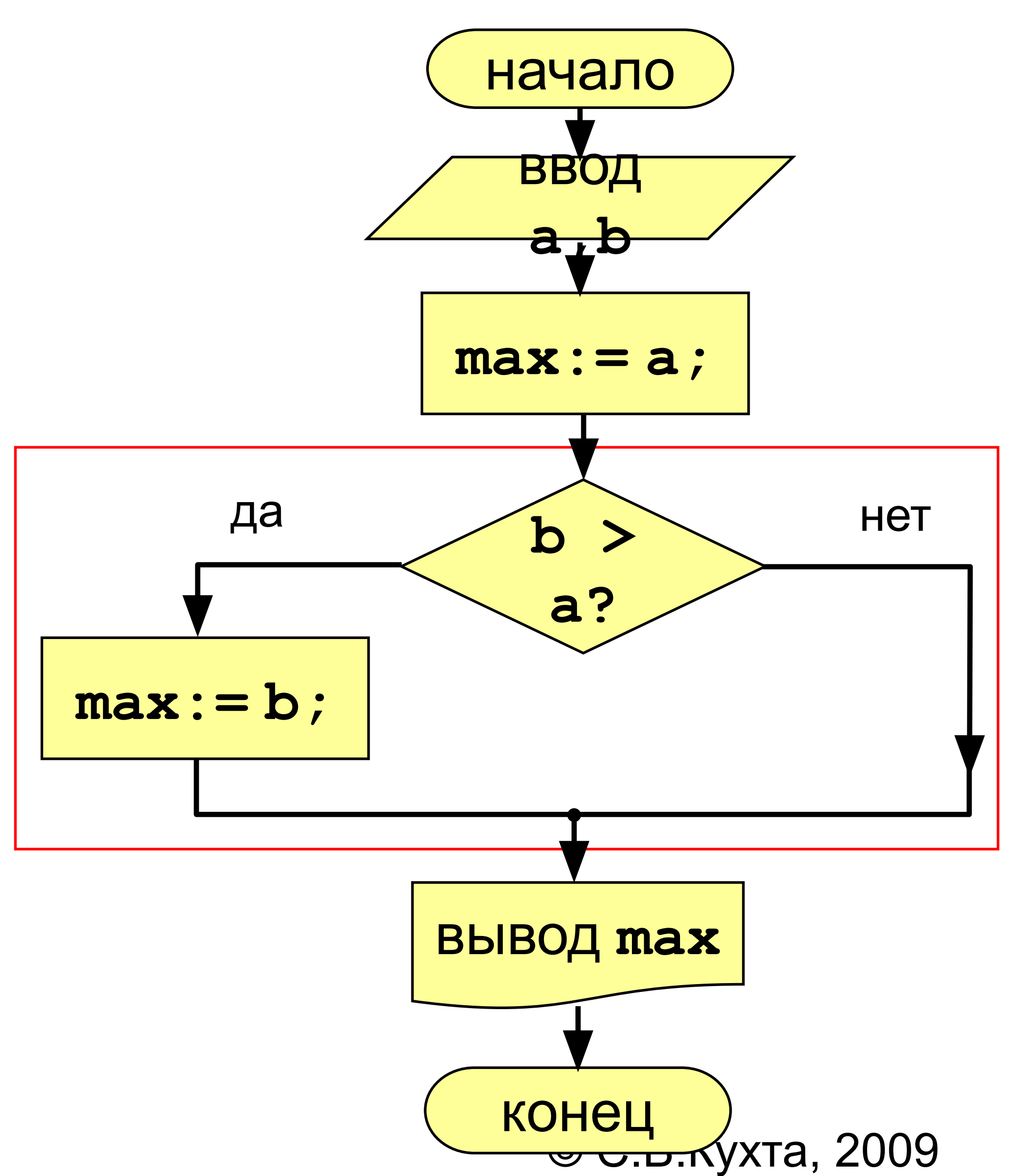
```
if a > b then begin
  a := b; end
else begin
  b := a;
end;
```

```
if a > b then begin
  a := b;
end
else begin
  b := a;
end;
```

```
if a > b then begin
  a := b;
end
else begin
  b := a;
end;
```

© С.В.Кухта, 2009

Вариант 2. Блок-схема



неполная форма ветвления

© С.В.Кухта, 2009

Вариант 2. Программа

```
program qq;
var a, b, max: integer;
begin
   writeln('Введите два целых числа');
   read ( a, b );
   max := a;
   if b > a then
      max := b;
   writeln ('Наибольшее число ', max);
end.
```

неполная
форма
условного
оператора

© С.В.Кухта, 2009

Вариант 2Б. Программа

```
program qq;
var a, b, max: integer;
begin
   writeln('Введите два целых числа');
   read ( a, b );
   max := b;
   if  a > b  then
         max := a;
   writeln ('Наибольшее число ', max);
end.
```

© С.В.Кухта, 2009

Что неправильно?

```
if a > b then
     a := b
else b := a;
```

```
if a > b then begin
   a := b;
end
else b := a;
```

```
if a > b then
     a := b
else b := a;
```

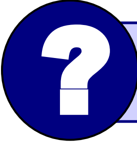
```
if b >= a then
   b := a;
```

© С.В.Кухта, 2009

Сложные условия

Задача. Фирма набирает сотрудников от 25 до 40 лет включительно. Ввести возраст человека и определить, подходит ли он фирме (вывести ответ «подходит» или «не подходит»).

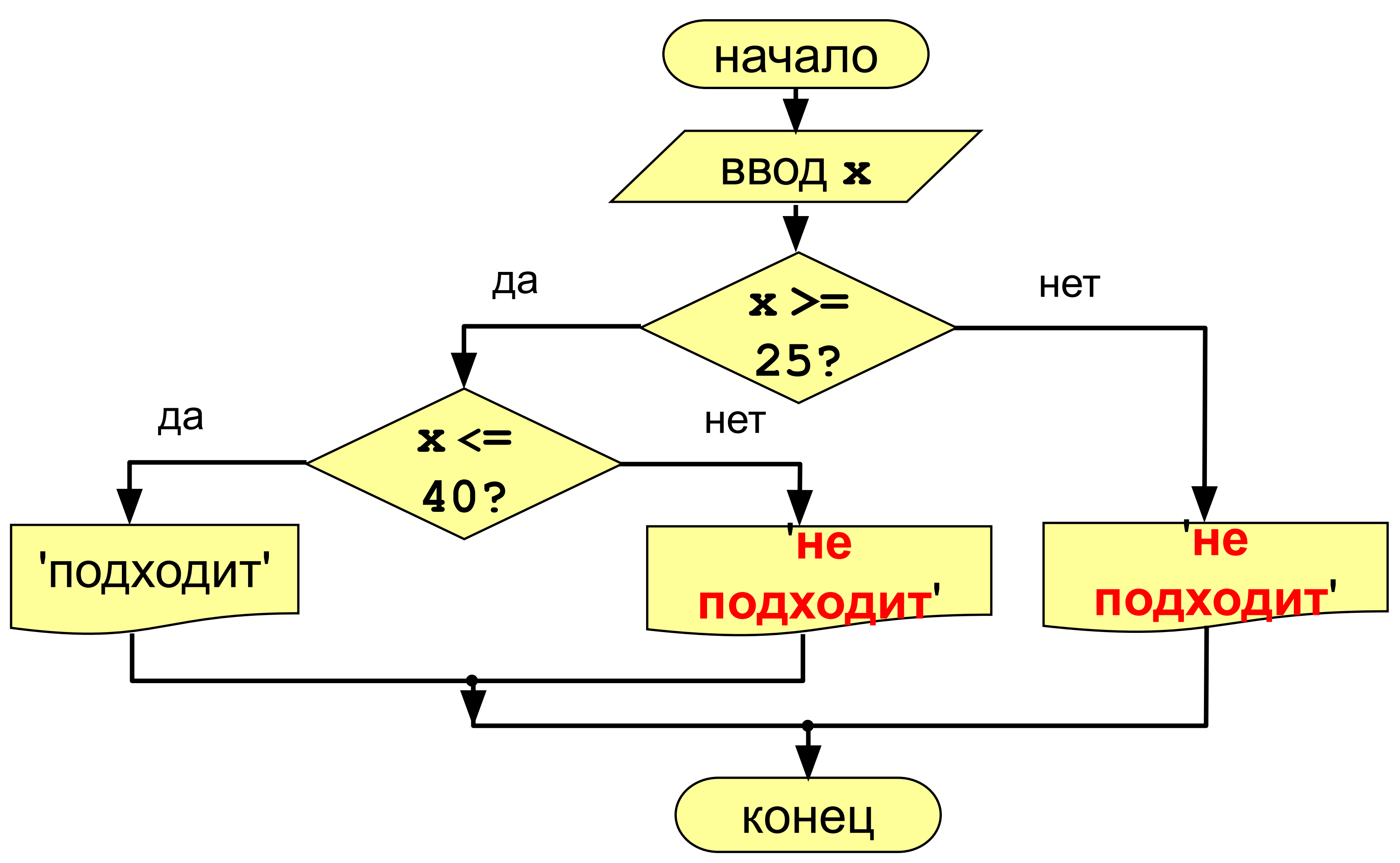
Особенность: надо проверить, выполняются ли два условия одновременно.



Можно ли решить известными методами?

© С.В.Кухта, 2009

Вариант 1. Алгоритм

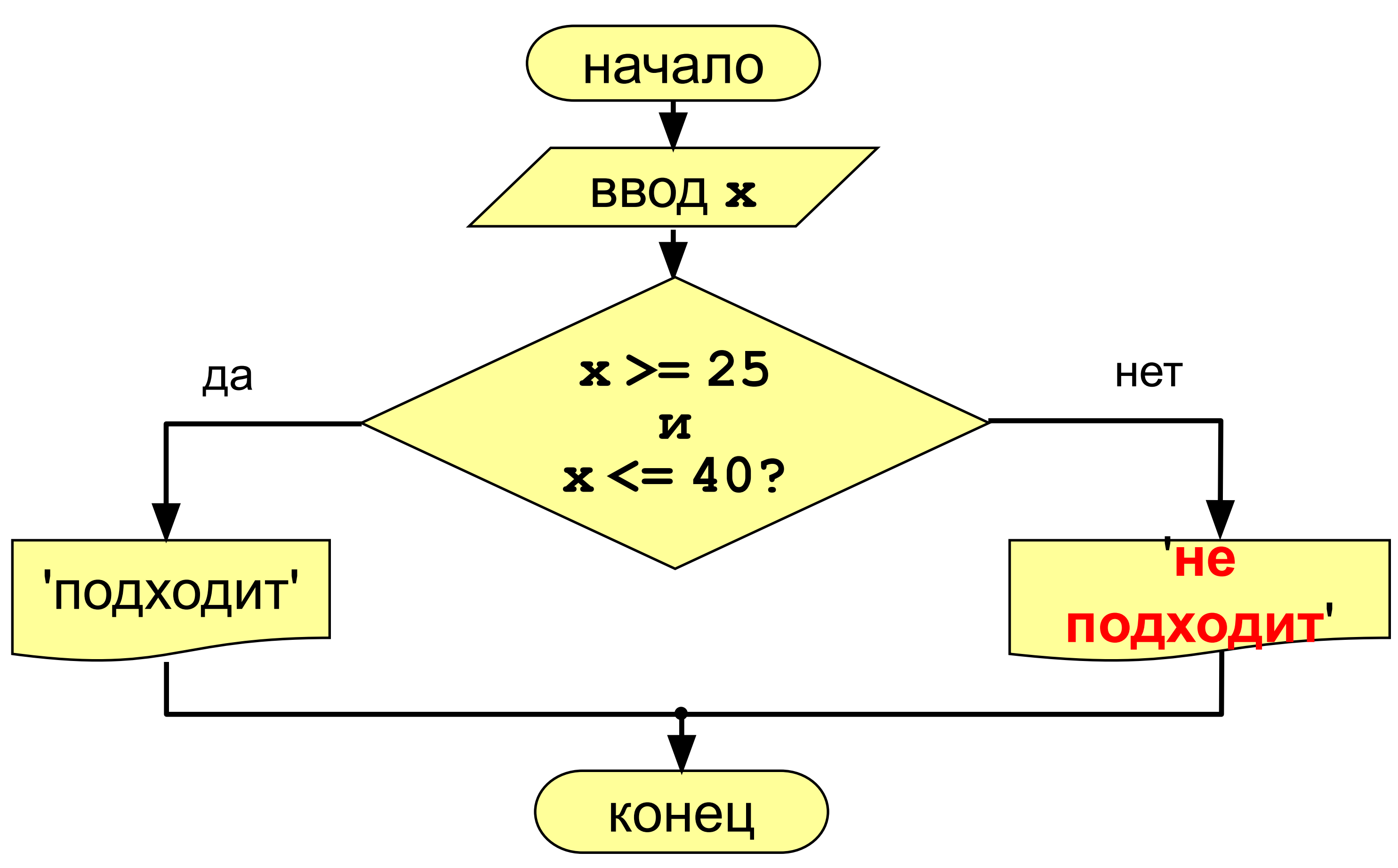


© С.В.Кухта, 2009

Вариант 1. Программа

```
program qq;
var x: integer;
begin
   writeln('Введите возраст');
   read ( x );
   if x >= 25 then
      if x <= 40 then
           writeln ('Подходит')
      else writeln ('Не подходит')
   else
      writeln ('Не подходит');
end.
```

Вариант 2. Алгоритм



© С.В.Кухта, 2009

Вариант 2. Программа

```
program qq;
var x: integer;
begin
  writeln('Введите возраст');
  read ( x );
  if (x >= 25) and (x <= 40) then
       writeln ('Подходит')
  else writeln ('Не подходит')
end.
```

сложное условие

© С.В.Кухта, 2009

Сложные условия

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью **логических операций**:

- **`not`** – НЕ (отрицание, инверсия)
- **`and`** – И (логическое умножение, конъюнкция, одновременное выполнение условий)
- **`or`** – ИЛИ (логическое сложение, дизъюнкция, выполнение хотя бы одного из условий)
- **`xor`** – исключающее ИЛИ (выполнение только одного из двух условий, но не обоих)

Простые условия (отношения)

равно

не равно

<   <=   >   >=   =   <>

© С.В.Кухта, 2009

Сложные условия

Порядок выполнения (приоритет = старшинство**)**

- выражения в скобках
- `not`
- `and`
- `or, xor`
- `<, <=, >, >=, =, <>`

Особенность – каждое из простых условий обязательно заключать в скобки.

Пример

```
       4      1      6     2        5
if not (a > b) or (c <> d) and (b <> a)
then begin
  ...
end
```

Сложные условия

Истинно или ложно при `a := 2; b := 3; c := 4;`

```
not (a > b)
(a < b) and (b < c)
not (a >= b) or (c = d)
(a < c) or (b < c) and (b < a)
(a < b) xor not (b > c)
```

True

True

True

True

FALSE

Для каких значений x истинны условия:

Условие	Множество	Упрощение
(x < 6) and (x < 10)	(-∞, 6)	x < 6
(x < 6) and (x > 10)	∅	
(x > 6) and (x < 10)	(6, 10)	
(x > 6) and (x > 10)	(10, ∞)	x > 10
(x < 6) or (x < 10)	(-∞, 10)	x < 10
(x < 6) or (x > 10)	(-∞, 6) ∪ (10,∞)	
(x > 6) or (x < 10)	(-∞, ∞)	
(x > 6) or (x > 10)	(6, ∞)	x > 6

© С.В.Кухта, 2009

Оператор выбора *case*

Оператор ***case*** позволяет сделать выбор между несколькими вариантами.

Оператор варианта состоит

- из выражения, называемого ***селектором***,
- и ***списка операторов***, каждый из которых отмечен константой того же типа, что и селектор.

Селектор должен относиться только к порядковому типу данных, но не к типу ***longint***.

Селектор может быть переменной или выражением.

Список констант может задаваться как явным перечислением, так и интервалом или их объединением. Повторение констант не допускается.

Тип переключателя и типы всех констант должны быть совместимыми.

© С.В.Кухта, 2009

Оператор выбора *case*

Формат:

```
Case < выражение {селектор}>  of
   <список констант 1> : < оператор 1>;
      .   .  .
   < список констант K> : < оператор K>;
     [else   < оператор K+1> ]
end
```

© С.В.Кухта, 2009

Оператор выбора *case*

Выполнение оператора ***case*** происходит следующим образом:

1) вычисляется значение селектора;
2) полученный результат проверяется на принадлежность к тому или иному списку констант;
3) если такой список найден, то дальнейшие проверки уже не производятся, а выполняется оператор, соответствующий выбранной ветви, после чего управление передается оператору, следующему за ключевым словом ***end***, которое закрывает всю конструкцию ***case***;
4) если подходящего списка констант нет, то выполняется оператор, стоящий за ключевым словом ***else***; если ***else***-ветви нет, то не выполняется ничего.

© С.В.Кухта, 2009

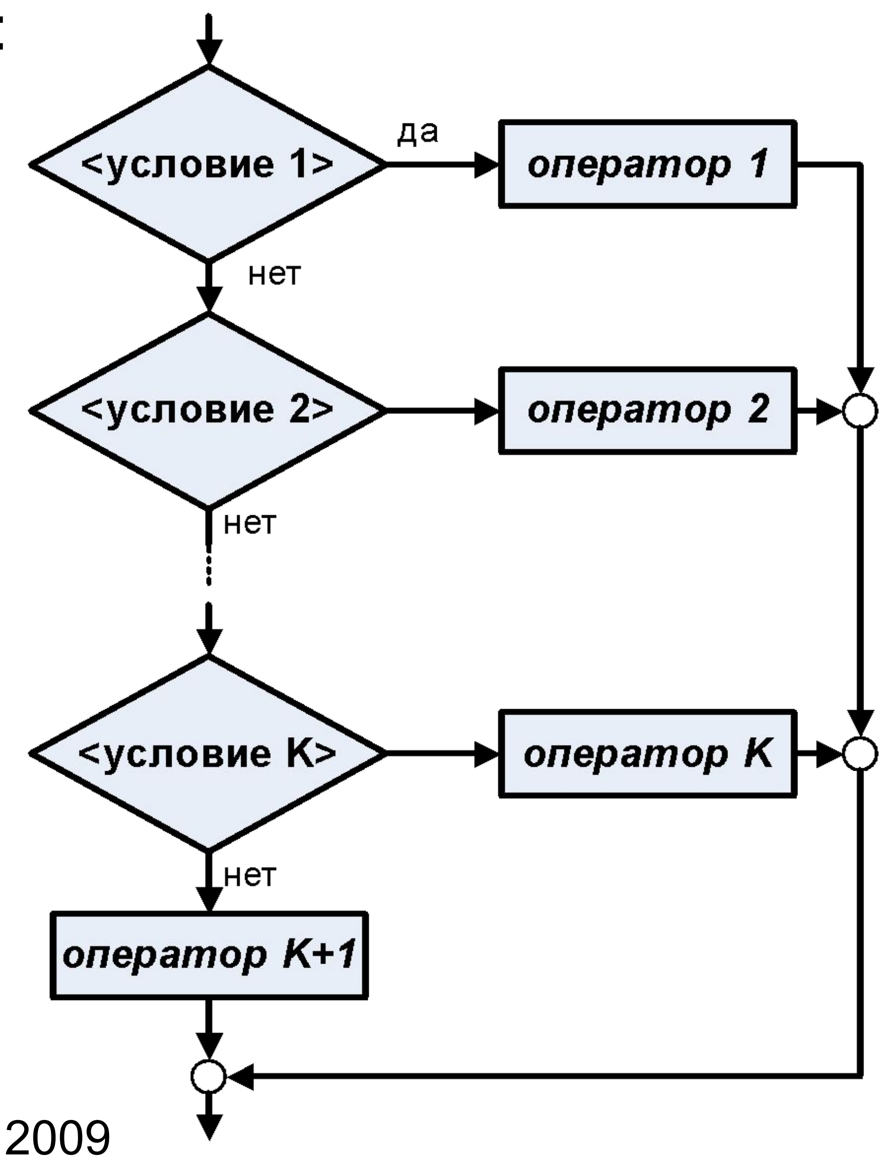
В операторе ***case*** по всем ветвям может выполняться только один оператор!

При необходимости выполнения нескольких требуется использовать операторные скобки ***begin-end***.

© С.В.Кухта, 2009

Оператор выбора *case*

Схема выполнения оператор:



© С.В.Кухта, 2009

Оператор выбора *case*

Примеры:

```
case Index mod 4 of
    0 :        x:= y*y + 1;
    1 :        x:= y*y - 2*y;
    2, 3 :  x:= 0
end;

case ch of
    'a','b','c' :    ch := succ(ch);
    'y','z' :      ch := pred(ch);
    'f','g' :      {пустой вариант};
    else      ch := pred(pred(ch))
end;
```

© С.В.Кухта, 2009

Оператор выбора *case*

Примеры:

```
case symbol(* :char *) of
  'a'..'z', 'A..'Z' : writeln('Это латинская буква');
  'а'..'я', 'А..'Я' : writeln('Это русская буква');
  '0'..'9' :    writeln('Это цифра');
  ' ',#10,#13,#26 :   writeln('Это пробельный символ');
  else                writeln('Это служебный символ');
end;
```

© С.В.Кухта, 2009

Оператор выбора

Задача: Ввести номер месяца и вывести количество дней в этом месяце.

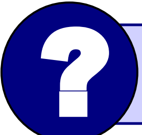
Решение: Число дней по месяцам:

28 дней – 2 (февраль)

30 дней – 4 (апрель), 6 (июнь), 9 (сентябрь), 11 (ноябрь)

31 день – 1 (январь), 3 (март), 5 (май), 7 (июль), 8 (август), 10 (октябрь), 12 (декабрь)

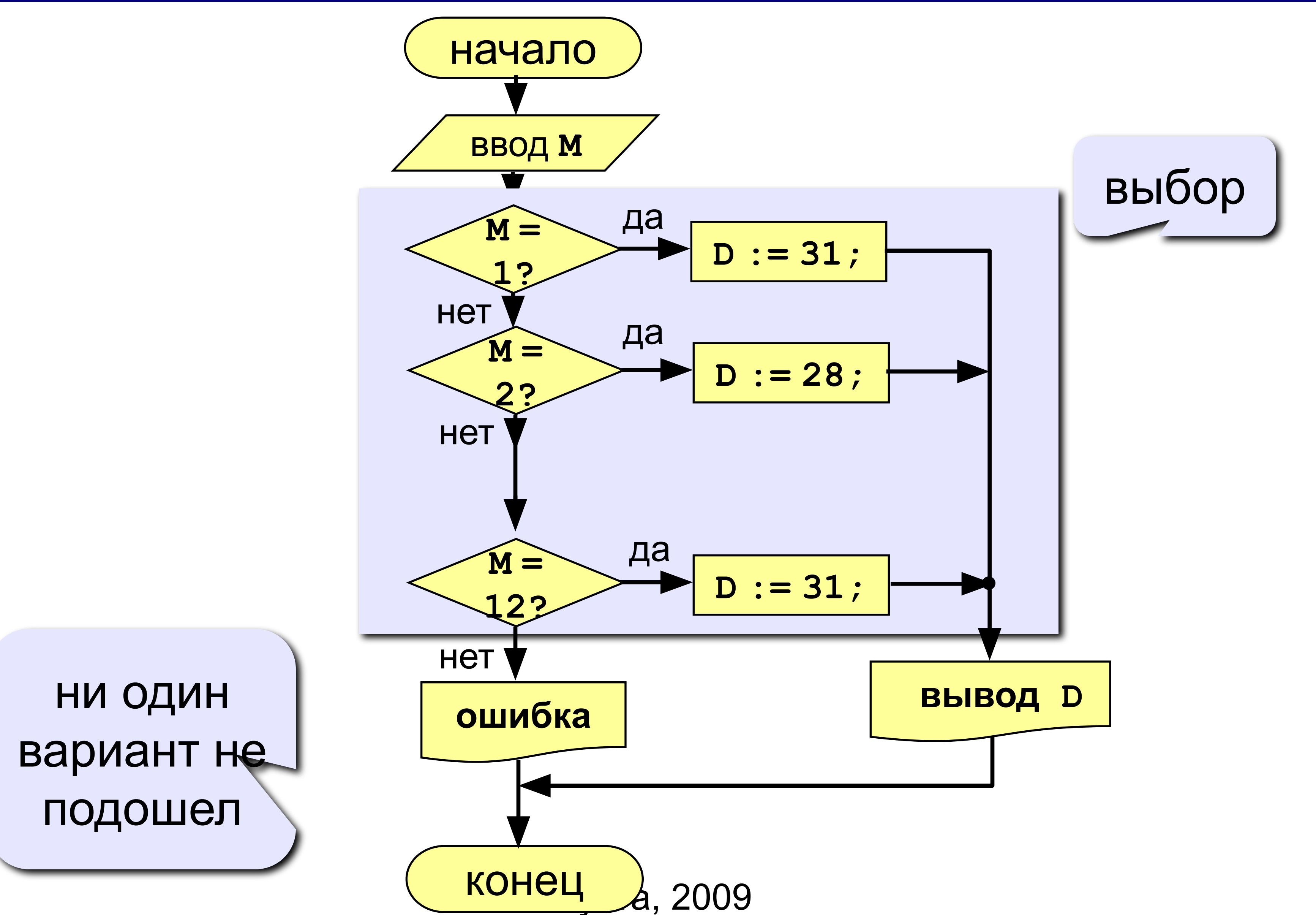
Особенность: Выбор не из двух, а из нескольких вариантов в зависимости от номера месяца.



Можно ли решить известными методами?

© С.В.Кухта, 2009

Алгоритм



Программа

```
program qq;
var M, D: integer;
begin
   writeln('Введите номер месяца:');
   read ( M );
   case M of
     2:          begin D := 28; end;
     4,6,9,11: begin D := 30; end;
     1,3,5,7,8,10,12: D := 31;
     else              D := -1;
   end;
   if D > 0 then
      writeln('В этом месяце ', D, ' дней.')
   else
      writeln('Неверный номер месяца');
end.
```

ни один вариант не подошел

© С.В.Кухта, 2009

Оператор выбора

Особенности:

- после **case** может быть имя переменной или арифметическое выражение целого типа (**integer**)

```
case i+3 of
 1: begin a := b; end;
 2: begin a := c; end;
end;
```

или символьного типа (**char**)

```
var c: char;
...
case c of
 'a': writeln('Антилопа');
 'б': writeln('Барсук');
 else writeln('Не знаю');
end;
```

Оператор выбора

Особенности:

- если нужно выполнить только один оператор, слова **begin** и **end** можно не писать

```
case i+3 of
 1: a := b;
 2: a := c;
end;
```

- нельзя ставить два одинаковых значения

```
case i+3 of
 1: a := b;
 1: a := c;
end;
```

© С.В.Кухта, 2009

Оператор выбора

Особенности:

- значения, при которых выполняются одинаковые действия, можно группировать

перечисление

диапазон

смесь

```
case i of
 1:              a := b;
 2,4,6:          a := c;
 10..15:         a := d;
 20,21,25..30:   a := e;
 else writeln('Ошибка');
end;
```

© С.В.Кухта, 2009

Что неправильно?

```
case a of
 2: begin a := b;
 4: a := c;
end;
```

```
case a of
 2: a := b ;
 4: a := c
end;
```

```
case a of
 2..5: a := b;
 4: a := c;
end;
```

```
case a of
 0..2: a := b;
 3..6: a := c;
end;
```

```
case a+c/2 of
 2: a := b;
 4: a := c;
end;
```

```
case a of
 2: begin a := b; d := 0; end;
 4: a := c;
end;
```

© С.В.Кухта, 2009

Иллюстрация *if* и *case*

В качестве примера, иллюстрирующего использование операторов ветвления, приведем несколько различных реализаций функции **sgn(x)** - знак числа **x**.

Из математики известно, что эта функция имеет следующие значения:

$$sgn(x) = \begin{cases} -1, & \text{если } x < 0; \\ 0, & \text{если } x = 0; \\ 1, & \text{если } x > 0. \end{cases}$$

© С.В.Кухта, 2009

Иллюстрация *if* и *case*

Реализовать эту функцию для случая, когда ***x*** вещественное, можно следующими способами (при условии, что `x:real; sgn: -1..1;`):

```
if x=0 then sgn:= 0;
if x<0 then sgn:= -1;
if x>0 then sgn:= 1;
```

Это так называемая реализация "в лоб". Здесь нет никаких хитростей и никаких попыток оптимизации:

- даже если сработает первый вариант, второй и третий все равно будут проверены, невзирая на то, что результат уже получен.

© С.В.Кухта, 2009

Иллюстрация *if* и *case*

```
if x=0 then sgn:= 0
else  if x<0 then sgn:= -1
            else sgn:= 1;
```

Этот вариант свободен от излишних проверок в случае, если значение переменной не положительно.

Эту реализацию следует признать более эффективной, чем предыдущая.

© С.В.Кухта, 2009

Иллюстрация *if* и *case*

```
if x=0 then sgn:=0
else  sgn:=x/abs(x);
```

Еще одна попытка сократить текст программы. Здесь используется стандартная функция ***abs()***, которая возвращает абсолютное значение аргумента.

Проблема в данном случае состоит в том, что "/" - деление дробное, но ведь нам необходим целый, а не вещественный ответ!

Можно воспользоваться стандартной функцией округления:

```
if x=0 then sgn:=0
else  sgn:=round(x/abs(x));
```

© С.В.Кухта, 2009

Иллюстрация *if* и *case*

```
case x=0 of
  true:  sgn:=0;
  false:    sgn:=round(x/abs(x));
end;
```

Пришлось заменить "***x***" на "***x = 0***".

Эта операция сравнения выдает результат логического типа ***boolean***, и именно логические константы ***true*** и ***false*** фигурируют в качестве меток выбора.

© С.В.Кухта, 2009

3. Массивы

© С.В.Кухта, 2009

Теперь мы приступаем к изучению массива - наиболее широко используемого структурированного типа данных, предназначенного для хранения нескольких однотипных элементов.

Массив – это последовательность однотипных данных, объединенная общим именем, элементы (компоненты) которой отличаются (идентифицируются) индексами.

Индекс элемента указывает место (номер) элемента в массиве.

Количество элементов массива фиксировано и определено в его описании. К элементам массива можно обращаться только по их номеру (индексу). Все компоненты массива являются одинаково доступными. Значения элементам массива присваиваются также как и другим переменным с учетом типа массива.

© С.В.Кухта, 2009

Массивы

Массив – это группа однотипных элементов, имеющих общее имя и расположенных в памяти рядом.

Особенности:

- все элементы имеют **один тип**
- весь массив имеет **одно имя**
- все элементы расположены в памяти **рядом**

Примеры:

- список студентов в группе
- квартиры в доме
- школы в городе
- данные о температуре воздуха за год

© С.В.Кухта, 2009

Массивы

A массив

1	2	3	4	5
5	10	15	20	25

НОМЕР элемента массива (**ИНДЕКС**)

A[1] A[2] ... A[5]

ЗНАЧЕНИЕ элемента массива

НОМЕР (ИНДЕКС) элемента массива: 2

ЗНАЧЕНИЕ элемента массива: 10

© С.В.Кухта, 2009

Объявление массивов

Зачем объявлять?

- определить **имя** массива
- определить **тип** массива
- определить **число элементов**
- выделить **место в памяти**

Массив целых чисел:


```
var A : array[ 1 .. 5 ] of integer ;
```

Размер через константу:

```
const N=5;
var A: array[1..N] of integer;
```

Объявление массивов

Массивы других типов:

```
var X, Y: array [1..10] of real;
    C: array [1..20] of char;
```

Другой диапазон индексов:

```
var Q: array [0..9] of real;
    C: array [-5..13] of char;
```

Ин

```
var A: array ['A'..'Z'] of real;
    B: array [False..True] of integer;
...
   A['C'] := 3.14259*A['B'];
   B[False] := B[False] + 1;
```

Объявление массива

Для того чтобы задать массив, необходимо в разделе описания переменных (*var*) указать его размеры и тип его компонент.

Общий вид описания (одномерного) массива:

```
array [<тип_индексов>] of <тип_компонент>;
```

Чаще всего это трактуется так:

```
array [<левая_граница>..<правая_граница>]
      of <тип_компонент>;
```

© С.В.Кухта, 2009

Нумерация

Нумеровать компоненты массива можно не только целыми числами.

Любой порядковый тип данных (перечислимый, интервальный, символьный, логический, а также произвольный тип, созданный на их основе) имеет право выступать в роли нумератора.

Таким образом, допустимы следующие описания массивов:

```
type charrr = 'a','c'..'z';
    (- отсутствует символ "b")
var a1: array[charrr] of integer;
        - 25 компонент
a2: array [char] of integer;
        - 256 целых компонент
a3: array [shortint] of real;
        - 256 вещественных компонент
```

Нумерация

Общий размер массива не должен превосходить 65 520 байт. Следовательно, попытка задать массив

```
a4:array[integer] of byte;
```

не увенчается успехом, поскольку тип ***integer*** покрывает 65 535 различных элементов.

А про тип ***longint*** в данном случае лучше и вовсе не вспоминать.

© С.В.Кухта, 2009

Тип компонент

Тип компонент массива может быть любым:

```
var a4: array[10..20] of real;
```
- массив из компонент простого типа

```
a5: array[0..100] of record1;
```
- массив из записей

```
a6: array[-10..10] of ^string;
```
- массив из указателей на строки

```
a7: array[-1..1] of file;
```
- массив из имен файловых переменных

```
a8: array[1..100] of array[1..100] of char;
```
- двумерный массив (массив векторов)

© С.В.Кухта, 2009

Что неправильно?

```
var a: array [1..10] of integer;
...
  A[5] := 4.5;
```

```
var a: array ['a'..'z'] of integer;
...
  A['b'] := 15;
```

```
var a: array [0..9] of integer;
...
  A[10] := 'X';
```

© С.В.Кухта, 2009

Массивы

Объявление:

```
const N = 5;
var a: array[1..N] of integer;
    i: integer;
```

Ввод с клавиатуры:

```
for i:=1 to N do begin
  write('a[', i, ']=');
  read ( a[i] );
end;
```

```
a[1] = 5
a[2] = 12
a[3] = 34
a[4] = 56
a[5] = 13
```

По

```
for i:=1 to N do a[i]:=a[i]*2;
```

Вы

```
writeln('Массив A:');
for i:=1 to N do
  write(a[i]:4);
```

```
Массив A:
  10  24  68 112  26
```

Многомерные массивы

Для краткости и удобства ***многомерные массивы*** можно описывать и более простым способом:

```
var a9: array[1..10,1..20] of real;
```
- двумерный массив 10 x 20

```
a10: array[boolean, -1..1, char, -10..10]
     of word;
```
- четырехмерный массив 2 x 3 x 256 x 21

Общее ограничение на размер массива - не более 65 520 байт - сохраняется и для многомерных массивов.

Количество компонент многомерного массива вычисляется как произведение всех его "измерений".

Таким образом, в массиве ***a9*** содержится 200 компонент, а в массиве ***a10*** - 32 256 компонент.

© С.В.Кухта, 2009

Описание переменных размерностей

Если ваша программа должна обрабатывать матрицы переменных размерностей (***N*** по горизонтали и ***M*** по вертикали), то вы столкнетесь с проблемой изначального задания массива, ведь в разделе ***var*** не допускается использование переменных.

Следовательно, самый логичный, казалось бы, вариант

```
var  m, n: integer;
  a: array[1..m, 1..n] of real;
```

придется отбросить.

© С.В.Кухта, 2009

Описание переменных размерностей

Если на этапе написания программы ничего нельзя сказать о предполагаемом размере входных данных, то не остается ничего другого, как воспользоваться техникой динамически распределяемой памяти (рассматривается во 2-м семестре).

© С.В.Кухта, 2009

Описание переменных размерностей

!

Предположим, однако, что известны максимальные границы, в которые могут попасть индексы обрабатываемого массива. Скажем, ***N*** и ***M*** заведомо не могут превосходить 100.

Тогда можно выделить место под наибольший возможный массив, а реально работать только с малой его частью:

```
const nnn=100;
var  a: array[1..nnn,1..nnn] of real;
  m, n: integer;
```

© С.В.Кухта, 2009

Обращение к компонентам массива

Массивы относятся к структурам прямого доступа. Это означает, что возможно напрямую (не перебирая предварительно все предшествующие компоненты) обратиться к любой интересующей нас компоненте массива.

Доступ к компонентам линейного массива осуществляется так:

```
<имя_массива>[<индекс_компоненты>]
```

а многомерного – так:

```
<имя_массива>[ <индекс 1>, ..., <индекс K>]
```

© С.В.Кухта, 2009

Обращение к компонентам массива

Массив состоит из элементов, имеющих порядковые номера, т.е. элементы массива упорядочены. Таким образом, если объекты одного типа обозначить именем, например "***A***", то элементы объекта будут ***A[1]***, ***A[2]*** и т.д. В квадратных скобках указан номер элемента.

Порядковый номер элемента массива, обычно не несет никакой информации о значении элемента, а показывает расположение элемента среди других.

© С.В.Кухта, 2009

Обращение к компонентам массива

Правила употребления ***индексов*** при обращении к компонентам массива:

1) Индекс компоненты может быть константой, переменной или выражением, куда входят операции и вызовы функций.
2) Тип каждого индекса должен быть совместим с типом, объявленным в описании массива именно для соответствующего "измерения"; менять индексы местами нельзя.
3) Количество индексов не должно превышать количество "измерений" массива. Попытка обратиться к линейному массиву как к многомерному обязательно вызовет ошибку. А вот обратная ситуация вполне возможна: например, если вы описали ***N***-мерный массив, то его можно воспринимать как линейный массив, состоящий из (***N***-1)-мерных массивов.

© С.В.Кухта, 2009

Описание переменных размерностей

Примеры использования компонент массива:

```
a1[1, 3]:= 0;
a1[i, 2]:= a1[i, 2]-1;
a2['z']:= a2['z']+1;
a3[-10]:= 2.5;
a3[i+j]:= a9[i, j];
a10[x>0, sgn(x), '!', abs(k*5)]:= 0;
```

© С.В.Кухта, 2009

Задание массива константой

Чтобы не вводить массивы вручную во время отладки программы, можно пользоваться не только файлами. Существует и более простой способ, когда входные данные задаются прямо в тексте программы при помощи типизированных констант.

Если массив линейный (вектор), то начальные значения для компонент этого вектора задаются через запятую, а сам вектор заключается в круглые скобки.

© С.В.Кухта, 2009

Задание массива константой

Многомерный массив также можно рассматривать как линейный, предполагая, что его компонентами служат другие массивы. Т.о., для системы вложенных векторов действует то же правило задания типизированной константы: каждый вектор ограничивается снаружи круглыми скобками.

Исключение составляют только массивы, компонентами которых являются величины типа ***char***. Такие массивы можно задавать проще: строкой символов.

© С.В.Кухта, 2009

Задание массива константой

Примеры задания массивов типизированными константами:

```
type mass = array[1..3, 1..2] of byte;
const a: array[-1..1] of byte = (0,0,0);
        {линейный}
  b: mass = ((1, 2),(3, 4),(5, 6));
        {двумерный}
  s: array[0..9] of char = '0123456789';
```

© С.В.Кухта, 2009

4. Цикл типа счетчик

© С.В.Кухта, 2009

Виды циклов

Цикл – основное средство в программировании.

Цикл – это последовательность операторов, которая может выполняться более одного раза.

Для реализации циклических алгоритмов в языке Паскаль используются операторы повторения:

- ***оператор цикла с параметром*** (типа счетчик);
- ***оператор цикла с предусловием***;
- ***оператор цикла с постусловием***.

Если количество повторов известно заранее, используется оператор цикла с параметром (типа счетчик).

Если количество повторов неизвестно, а задано некоторое условие окончания или продолжения цикла применяются операторы цикла с предусловием или оператор цикла с постусловием.

© С.В.Кухта, 2009

Цикл *for*

В случае когда количество однотипных действий заранее известно (например, необходимо обработать все компоненты массива), стоит отдать предпочтение циклу с параметром (***for***).

Оператор ***for*** предусматривает повторное выполнение некоторого оператора с одновременным изменением значения, присваиваемого управляющей переменной (параметру этого цикла).

© С.В.Кухта, 2009

Цикл *for*

Оператор ***For*** состоит из заголовка и тела цикла. Он может быть представлен в двух форматах:

```
for <имя> := N1 to N2 do
      <оператор>;
```

```
for <имя> := N1 downto N2 do
      <оператор>;
```

Декрементный цикл с параметром

Инкрементный цикл с параметром

Здесь ***for … do*** – заголовок цикла; **<имя>** – это имя переменной – параметра цикла; ***N1*** – ее начальное значение; ***N2*** – ее конечное значение; ***<оператор>*** – тело цикла.

Тело цикла может быть простым или составным оператором.

© С.В.Кухта, 2009

Цикл *for*

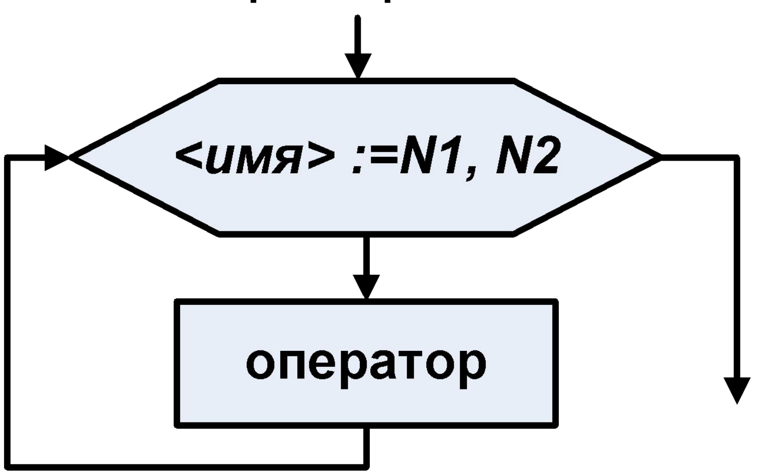
Переменная цикла (счетчик), нижняя граница ***N1*** (переменная, константа или выражение) и верхняя граница ***N2*** (переменная, константа или выражение) должны относиться к эквивалентным порядковым типам данных.

Если тип нижней или верхней границы не эквивалентен типу счетчика, а лишь совместим с ним, то осуществляется неявное приведение: значение границы преобразуется к типу счетчика, в результате чего возможны ошибки.

© С.В.Кухта, 2009

Цикл *for*

Схема выполнения оператора:



© С.В.Кухта, 2009

Цикл *for*

Цикл ***for-to*** работает следующим образом:

1) вычисляется значение верхней границы ***N2***;
2) переменной цикла присваивается значение нижней границы ***N1***;
3) производится проверка того, что переменная цикла не превосходит ***N2***;
4) если это так, то выполняется ***<оператор>***;
5) значение переменной цикла увеличивается на единицу;
6) пункты 3-5, составляющие одну итерацию цикла, выполняются до тех пор, пока переменная цикла не станет строго больше, чем ***N2***; как только это произошло, выполнение цикла прекращается, а управление передается следующему за ним оператору.

© С.В.Кухта, 2009

Цикл *for*

Из этой последовательности действий можно понять, какое количество раз отработает цикл ***for-to*** в каждом из трех случаев:

- ***N1*** < ***N2***: цикл будет работать ***N2-N1***+1 раз;
- ***N1*** = ***N2***: цикл отработает ровно один раз;
- ***N1*** > ***N2***: цикл вообще не будет работать.

После окончания работы цикла переменная-счетчик может потерять свое значение. Таким образом, нельзя с уверенностью утверждать, что после того, как цикл завершил работу, обязательно окажется, что ее значение равно ***N2***+1.

Поэтому попытки использовать переменную-счетчик сразу после завершения цикла (без присваивания ей какого-либо нового значения) могут привести к непредсказуемому поведению программы при отладке.

© С.В.Кухта, 2009

Цикл *for*

Цикл ***for-downto*** работает следующим образом:

1) вычисляется значение верхней границы ***N2***;
2) переменной цикла присваивается значение нижней границы ***N1***;
3) производится проверка того, что переменная цикла не меньше ***N2***;
4) если это так, то выполняется ***<оператор>***;
5) значение переменной цикла уменьшается на единицу;
6) пункты 1-3 выполняются до тех пор, пока переменная цикла не станет меньше, чем ***N2***; как только это произошло, выполнение цикла прекращается, а управление передается следующему за ним оператору.

© С.В.Кухта, 2009

Цикл *for*

Из этой последовательности действий можно понять, какое количество раз отработает цикл ***for-downto*** в каждом из трех случаев:

- ***N1*** < ***N2***: цикл вообще не будет работать$
- ***N1*** = ***N2***: цикл отработает ровно один раз;
- ***N1*** > ***N2***: цикл будет работать ***N1***-***N2***+1 раз.

Замечание о неопределенности значения счетчика после окончания работы цикла справедливо и в этом случае.

© С.В.Кухта, 2009

Основные требования к оператору *FOR*

!

- Параметр цикла, ***начальное*** и ***конечное значения*** **должны быть** ***одного и того же порядкового типа*** (лучше всего – целого типа, недопустимо - вещественного).
- ***Начальное*** и ***конечное значения*** вычисляются лишь один раз – при входе в цикл, и, следовательно, **должны быть** ***определены до входа в цикл и не могут быть изменены в теле цикла***.
- Дополнительно (принудительно) ***изменять значение параметра в теле цикла не рекомендуется***, поскольку контроль за правильностью исполнения такого цикла очень затруднен.
- Не допускается изменение параметра цикла на величину, отличную от единицы.

© С.В.Кухта, 2009

Цикл *for*

Примеры:

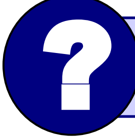
```
For i:=1 to 20 do
  writeln(Sqrt(i));
  {выведет 20 результатов извлечения
      квадратного корня из i  }
For  ch:='A'  to 'z' do
  writeln(ch);
  {выведет латинские буквы }
For  j:=14 downto 10 do
  writeln(j);
  {выведет числа от 14 до 10}
```

© С.В.Кухта, 2009

Цикл *for*

Задача. Вывести на экран квадраты и кубы целых чисел от 1 до 8 (от **`a`** до **`b`**).

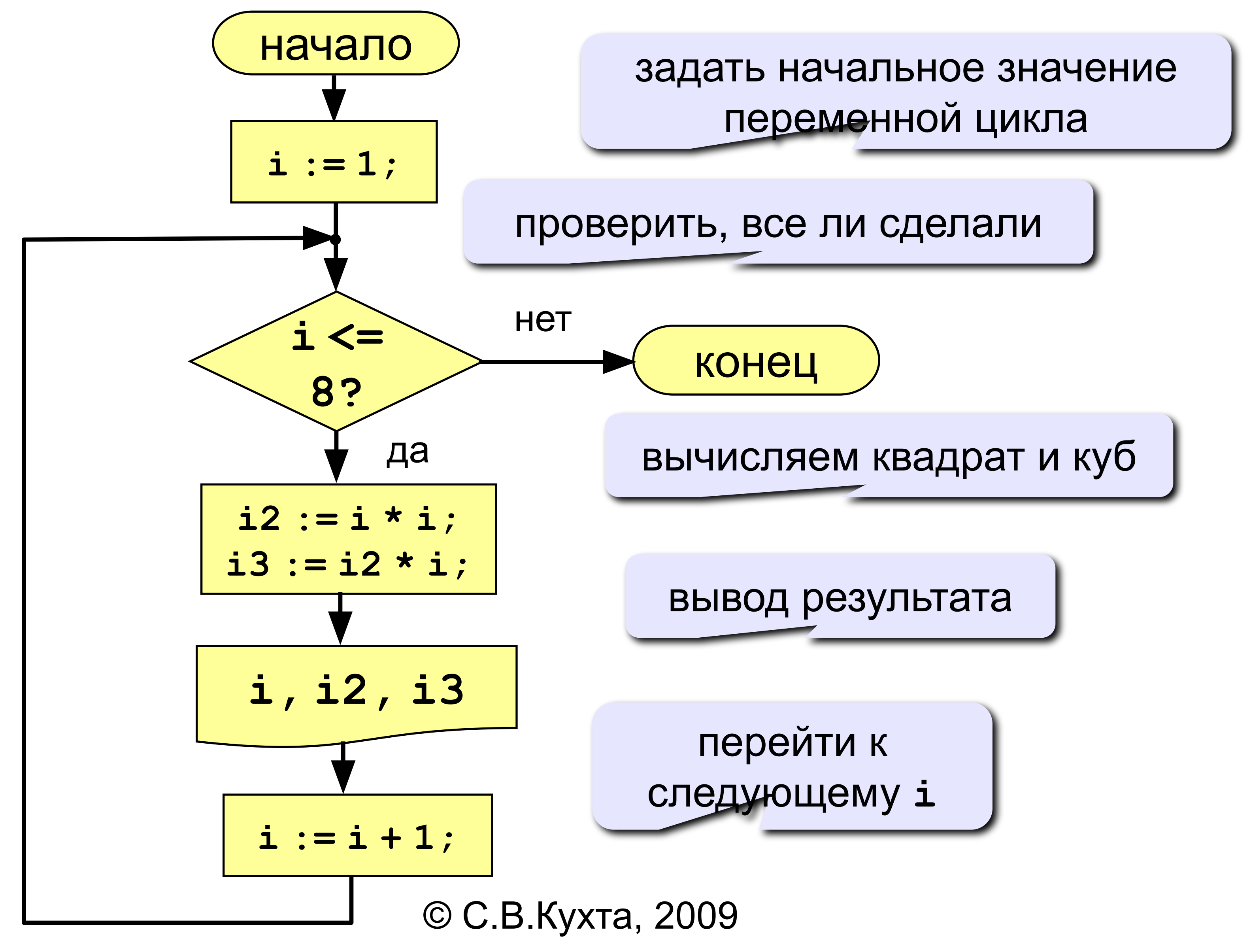
Особенность: одинаковые действия выполняются 8 раз.



Можно ли решить известными методами?

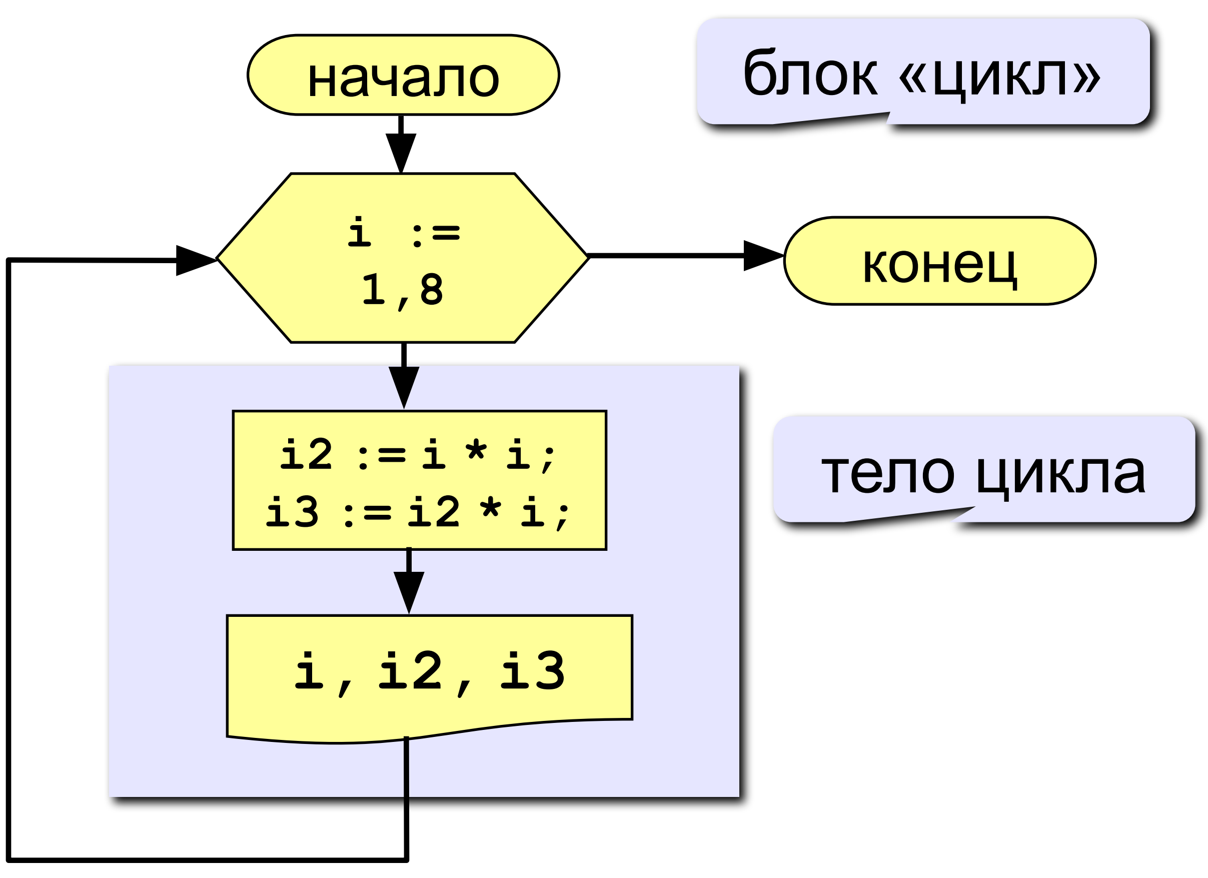
© С.В.Кухта, 2009

Алгоритм



© С.В.Кухта, 2009

Алгоритм (с блоком «цикл»)



© С.В.Кухта, 2009

Программа

```
program qq;
var i, i2, i3: integer;
begin
  for i:=1 to 8 do begin
    i2 := i*i;
    i3 := i2*i;
    writeln(i:4, i2:4, i3:4);
  end;
end.
```

переменная цикла

начальное значение

конечное значение

© С.В.Кухта, 2009

Цикл с уменьшением переменной

Задача. Вывести на экран квадраты и кубы целых чисел от 8 до 1 (в обратном порядке).

Особенность: переменная цикла должна уменьшаться.

Решение:

```
for i:=8 downto 1 do begin
    i2 := i*i;
    i3 := i2*i;
    writeln(i:4, i2:4, i3:4);
 end;
```

© С.В.Кухта, 2009

Цикл *for*

Особенности:

- после выполнения цикла **во многих системах** устанавливается первое значение переменной цикла, при котором нарушено условие:

```
for i:=1 to 8 do
  writeln('Привет');
writeln('i=', i);
```

i=9

НЕ ДОКУМЕНТИРОВАНО

```
for i:=8 downto 1 do
  writeln('Привет');
writeln('i=', i);
```

i=0

© С.В.Кухта, 2009

Сколько раз выполняется цикл?

```
a := 1;
for i:=1 to 3 do a := a+1;
```

a = 4

```
a := 1;
for i:=3 to 1 do a := a+1;
```

a = 1

```
a := 1;
for i:=1 downto 3 do a := a+1;
```

a = 1

```
a := 1;
for i:=3 downto 1 do a := a+1;
```

a = 4

© С.В.Кухта, 2009

Как изменить шаг?

Задача. Вывести на экран квадраты и кубы нечётных целых чисел от 1 до 9.

Особенность: переменная цикла должна увеличиваться на 2.

Проблема: в Паскале шаг может быть 1 или -1.

Решение:

```
for i:=1 to 9 do begin
    if i mod 2 = 1 then begin
      i2 := i*i;
      i3 := i2*i;
      writeln(i:4, i2:4, i3:4);
    end;
  end;
```

выполняется только для нечетных **i**

Что плохо?

Как изменить шаг? – II

Идея: Надо вывести всего 5 чисел, переменная **k** изменяется от 1 до 5. Начальное значение **i** равно 1, с каждым шагом цикла **i** увеличивается на 2.

Решение:

```
i := 1;
for k:=1 to 5 do begin
  i2 := i*i;
  i3 := i2*i;
  writeln(i:4, i2:4, i3:4);
  i := i + 2;
end;
```

© С.В.Кухта, 2009

Как изменить шаг? – III

Идея: Надо вывести всего 5 чисел, переменная **k** изменяется от 1 до 5. **Зная k, надо рассчитать i.**

k	1	2	3	4	5
i	1	3	5	7	9

i = 2k-1

Решение:

```
for k:=1 to 5 do begin
   i := 2*k - 1;
   i2 := i*i;
   i3 := i2*i;
   writeln(i:4, i2:4, i3:4);
 end;
```

© С.В.Кухта, 2009

Пример 1

Вычислить факториал числа ***n***. По определению факториалом натурального числа ***n*** (обозначается ***n!***) называется произведение чисел от 1 до ***n***:

$$n! = \prod_{k=1}^{n} k = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

```
Program  NFactorial;
var  Factorial, Argument: Integer;
     i : Integer;
Begin
    Write(' введите аргумент факториала ');
    Readln(Argument) ;
    Factorial := 1;
    For   i:=2  to  Argument do
          Factorial := i*Factorial;
    Writeln(Argument,'! = ', Factorial)
End.
```

Пример 2

Табулирование (таблица значений) функции $y=\exp(-x^2/2)$.

```
Program Tabulation;
var   MinBound, MaxBound, Step, x, y, Coef :Real;
   i, n : Integer;
Begin
     Write('Введите пределы табулирования ');
     Readln(MinBound, MaxBound);
     Write('Введите шаг табулирования ');
     Readln(Step);
     n := Round((MaxBound - MinBound)/Step);
     x := MinBound;
     for   i:=0   to   n   do  begin
       y := exp(-Sqr(x)/2);
       writeln(' x = ', x, ' y = ', y);
       x := x + Step
     end;
End.
```

Пример 3

Вычислить сумму **N** первых членов ряда $1^2+3^2+5^2+...+(2*N-1)^2$.

```
PROGRAM SUM_N;       { расчет конечной суммы }
var  a,  S, i,  N:  word;
Begin
     write('Введите число членов суммы N=');
     readln(N);
     S:= 0;
     For  i:=1 to N do begin  { цикл суммирования}
       a:= Sqr(2*i-1);
       S:= S+a
     end;
     Writeln('Конечная сумма S=',  S:10);
     Writeln('Нажми Enter');
     readln
End.
```

© С.В.Кухта, 2009

Пример 4

Программа вычисляет скалярное произведение вектора *V* и вектора *V**, полученного из *V* перестановкой координат в обратном порядке.

```
Program  ScalarMult;
 Const    n = 10;
 Type   Vector = array[1..n]  of  Real;
 Var  V : Vector;   Summa : Real;   i : Integer;
 Begin
 For i:=1 to n do begin  { блок ввода исходного вектора }
        Write('Введите координату вектора: ', i);
        Readln(V[i]);
    end;
    Summa := 0;  { блок вычислений }
    For i:=1 to n do
         Summa := Summa + V[i] * V[n-i+1];
    write(' Результат : ', Summa:2:4)
 End.
```

!

Пример 5

Программа вычисляет произведение матриц $A_{n×k}$ и $B_{k×m}$.

```
Program  MatrixMult;
Var   A, B, C : array [1..20, 1..20] of  Real;
   n, m, k, i , j, p: Integer;
Begin
     Write('Ведите размерности матриц : ');
     Readln (n, k, m);
```

{ блок ввода исходных матриц }

!

```
      For i:=1 to n do
         For j:=1 to k do begin
              Writeln('Введите элемент матрицы A[',
                            i,',',j,']');
                Read(A[i, j]);   end;
```

© С.В.Кухта, 2009

Пример 5

```
For i:=1 to k do
     For j:=1 to m do begin
        Writeln('Введите элемент матрицы B[', i,
                ',',j,']');
              Read(B[i, j]);   end;
 { блок умножения матриц }
   For i:=1 to n do
     For j:=1 to m do begin
           C[i, j]:= 0;
           For p:=1 to k do
            C[i, j]:=C[i, j]+A[i, p]*B[p, j]
     end;
```

© С.В.Кухта, 2009

Пример 5

{ блок вывода матрицы построчно }

```
For  i:=1 to n do begin
  Writeln;
  For j:=1 to m do
     Write(C[i, j]:8:2)
end;
```

```
End.
```

© С.В.Кухта, 2009

4. Циклы с условием

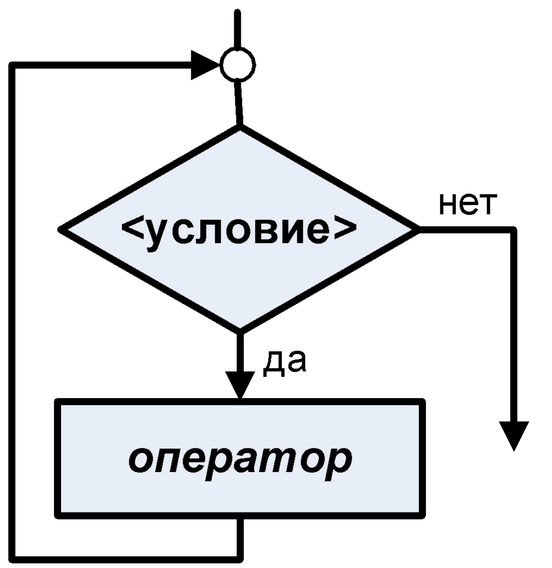
© С.В.Кухта, 2009

Цикл с предусловием

Формат:

```
while <условие> do begin
   {тело цикла}
end;
```

Схема выполнения оператора:



Последовательность действий при выполнении цикла:

1. Проверяется истинно ли <условие>.
2. Если это так, то выполняется операторы, заданные в теле цикла.
3. Пункты 1 и 2 выполняются до тех пор, пока <условие> не станет ложным.

© С.В.Кухта, 2009

Цикл с предусловием

Особенности:

- условие окончания цикла может быть выражено переменной, константой или выражением, имеющим логический тип.
- можно использовать сложные условия:

```
while (a<b) and (b<c) do begin
  {тело цикла}
end;
```

- если в теле цикла только один оператор, слова **begin** и **end** можно не писать:

```
while a < b do
   a := a + 1;
```

© С.В.Кухта, 2009

Цикл с предусловием

Особенности:

- условие пересчитывается **каждый раз** при входе в цикл
- если условие на входе в цикл ложно, цикл не выполняется ни разу

```
a := 4; b := 6;
while a > b do
   a := a - b;
```

<оператор>.

- если условие никогда не станет ложным, программа **зацикливается**

```
a := 4; b := 6;
while a < b do
   d := a + b;
```

Цикл с предусловием

Перед началом цикла должны быть инициализированы переменные, входящие в условие цикла.
Чтобы избежать «зацикливания», среди операторов тела цикла обязательно требуется предусмотреть изменение переменных, входящих в проверяемое условие.

© С.В.Кухта, 2009

Сколько раз выполняется цикл?

```
a := 4; b := 6;
while a < b do a := a + 1;
```

2 раза
a = 6

```
a := 4; b := 6;
while a < b do a := a + b;
```

1 раз
a = 10

```
a := 4; b := 6;
while a > b do a := a + 1;
```

0 раз
a = 4

```
a := 4; b := 6;
while a < b do b := a - b;
```

1 раз
b = -2

```
a := 4; b := 6;
while a < b do a := a - 1;
```

зацикливание

© С.В.Кухта, 2009

Цикл с неизвестным числом шагов

Пример: Определить количество цифр числа.

Задача: Ввести целое число (<2000000) и определить число цифр в нем.

Идея решения: Отсекаем последовательно последнюю цифру, увеличиваем счетчик.

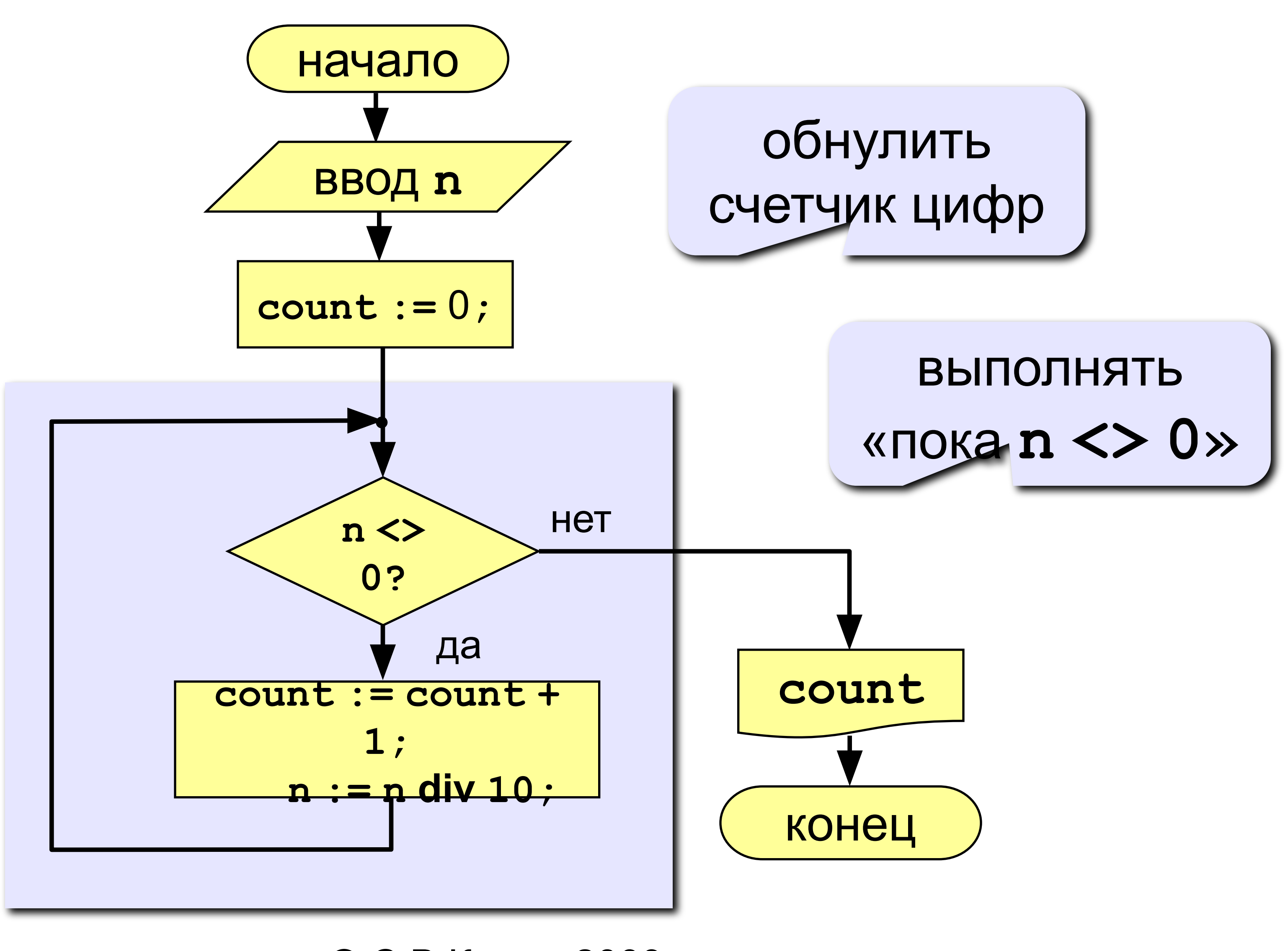
n	count
123	0
12	1
1	2
0	3

Проблема: Неизвестно, сколько шагов надо сделать.

Решение: Надо остановиться, когда **n = 0**, т.е. надо делать «пока **n <> 0**».

© С.В.Кухта, 2009

Алгоритм



© С.В.Кухта, 2009

Программа

```
program qq;
var n, count, n1: integer;
begin
  writeln('Введите целое число');
  read(n); n1 := n;
  count := 0;
   while n <> 0 do begin
      count := count + 1;
      n := n div 10;
    end;
  writeln('В числе ', n1, ' нашли ',
            count, ' цифр');
end.
```

выполнять «пока **n** <> 0»

значение n потеряно: печатается 0!

? **Что плохо?**

Замена for на while и наоборот

```
for i:=1 to 10 do begin
  {тело цикла}
end;
```

```
i := 1;
while i <= 10 do begin
  {тело цикла}
  i := i + 1;
end;
```

```
for i:=a downto b do
  begin
  {тело цикла}
end;
```

```
i := a;
while i >= b do begin
  {тело цикла}
  i := i - 1;
end;
```

Замена цикла for на while возможна всегда.

Замена while на for возможна только тогда, когда можно заранее рассчитать число шагов цикла.

Пример 1

Найти наименьшее натуральное решение неравенства $x^3+ax^2+bx+c > 0$ с целыми коэффициентами.

Идея решения: Очевидный алгоритм поиска решения заключается в последовательном вычислении значений $Y=x^3+ax^2+bx+c$ для $x = 1, 2, 3, ...$ до тех пор, пока $Y \le 0$.

© С.В.Кухта, 2009

Пример 1

```
Program UneqvSolution;
 Var  a, b, c, X : Integer; Y : Real;
 Begin
 Write('Введите коэффициенты a, b, c : ');
 Readln(a, b, c);
    X := 1;   Y :=a+b+c+1;    { Инициализация цикла }
    While  Y <= 0 do begin
         X := Succ(X);         { Следующее значение X }
    Y :=((X+a)*X+b)*X+c          { Следующее значение Y }
    end;
    Writeln('X = ', X, ' Y = ', Y )
End.
```

© С.В.Кухта, 2009

Пример 2: последовательности

Примеры:

- 1, 2, 3, 4, 5, …
- 1, 2, 4, 7, 11, 16, …
- 1, 2, 4, 8, 16, 32, …
- $\frac{1}{2}, \frac{1}{2}, \frac{3}{8}, \frac{1}{4}, \frac{5}{32}, \ldots$

$a_n = n$

$a_1 = 1, \quad a_{n+1} = a_n + 1$

$a_1 = 1, \quad a_{n+1} = a_n + n$

$a_n = 2^{n-1}$

$a_1 = 1, \quad a_{n+1} = 2a_n$

$\frac{1}{2}, \frac{2}{4}, \frac{3}{8}, \frac{4}{16}, \frac{5}{32}, \ldots$

$$a_n = \frac{b_n}{c_n}$$

$b_1 = 1, \quad b_{n+1} = b_n + 1$

$c_1 = 2, \quad c_{n+1} = 2c_n$

© С.В.Кухта, 2009

Пример 2: последовательности

Задача: найти сумму всех элементов последовательности,

$$1,\ -\frac{1}{2},\ \frac{2}{4},\ -\frac{3}{8},\ \frac{4}{16},\ -\frac{5}{32},\ldots$$

которые по модулю больше 0,001:

$$S = 1 - \frac{1}{2} + \frac{2}{4} - \frac{3}{8} + \frac{4}{16} - \frac{5}{32} + \ldots$$

Элемент последовательности (начиная с №2):

$$a = z\,\frac{b}{c}$$

n	1	2	3	4	5	...
b	1	2	3	4	5	...
c	2	4	8	16	32	...
z	-1	1	-1	1	-1	...

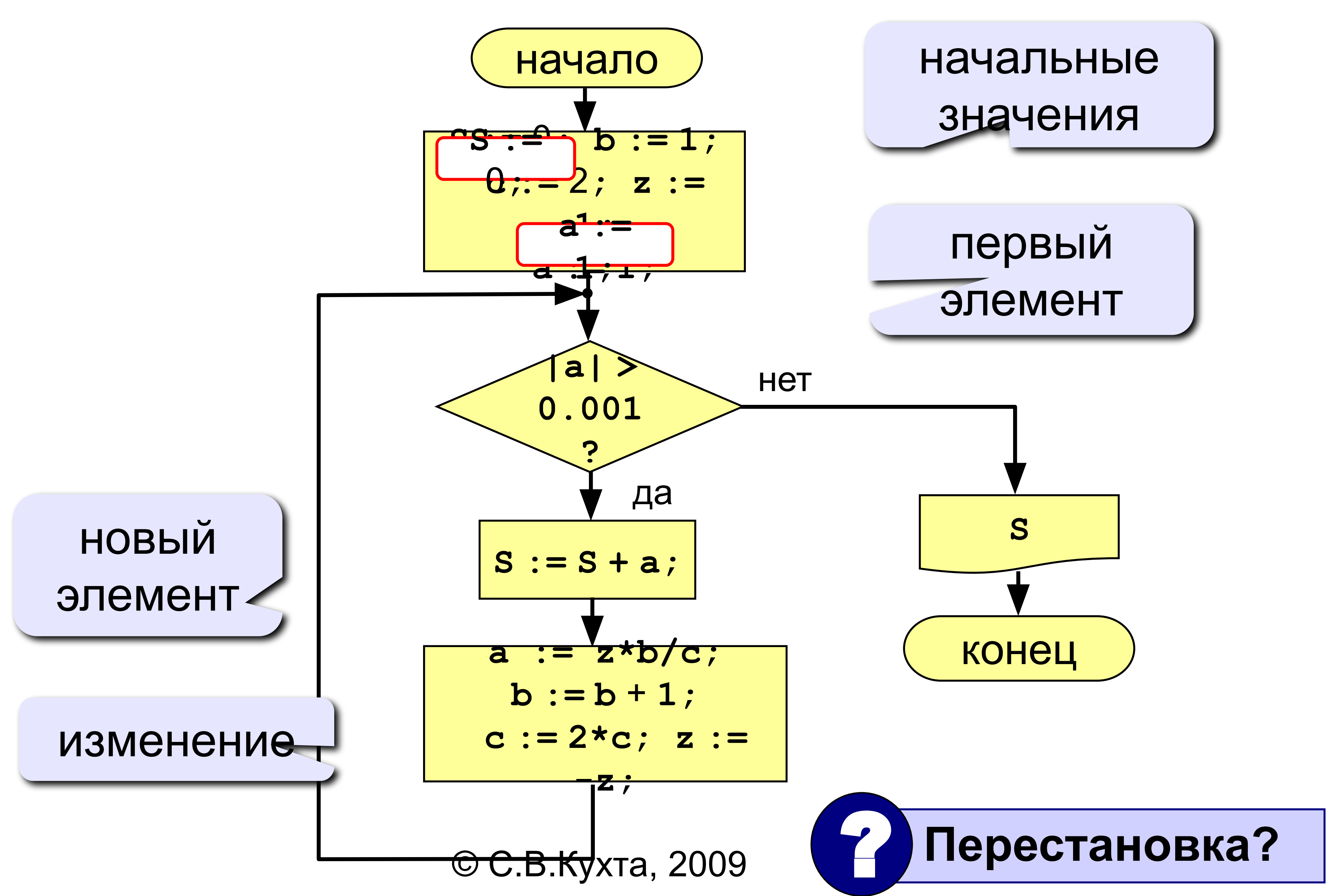
```
b := b+1;
```

```
c := 2*c;
```

```
z := -z;
```

© С.В.Кухта, 2009

Алгоритм



© С.В.Кухта, 2009

Программа

```
program qq;
var b, c, z: integer;
    S, a: real;
begin
   S := 0; z := -1;
   b := 1; c := 2; a := 1;
   while abs(a) > 0.001 do begin
      S := S + a;
      a := z * b / c;
      z := - z;
      b := b + 1;
      c := c * 2;
   end;
   writeln('S =', S:10:3);
end.
```

начальные значения

увеличение суммы

расчет элемента последовательности

переход к следующему слагаемому

© С.В.Кухта, 2009

Пример 3

С точностью ε=0,001 найти значения функции ***y=sin(x)*** при некотором значении ***x*** с использованием представления функции в виде ряда:

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n+1}\frac{x^{2n+1}}{(2n+1)!} + \ldots$$

© С.В.Кухта, 2009

Пример 3

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n+1}\frac{x^{2n+1}}{(2n+1)!} + \ldots$$

Идея решения. Пусть $a_1 = x$, $a_3 = x3/3!$, …

Легко заметить, что:

$x^3=x*x^2$; $x^5=x^3*x^2$; … $x^k=x^{k-2}*x^2$;

$1!=1$; $2!=1\cdot2=1!\cdot2$; $3!=1\cdot2\cdot3=1!\cdot2\cdot3$;

$5!=1\cdot2\cdot3\cdot4\cdot5=3!\cdot4\cdot5$;

$7!=5!\cdot6\cdot7$; …

$k!=(k-2)!\cdot(k-1)\cdot k$.

Т.о. получаем

$a_k = p * a_{k-2}$, где $p = (-x^2)/((k-1)*k)$, k= 3, 5, 7, ...

Т.е., каждый член ряда a_k при k>0 можно получить умножением предыдущего члена ряда a_{k-2} на коэффициент p.

© С.В.Кухта, 2009

Пример 3

Приближенное значение функции $y=sin(x)$ находится как частичная сумма k членов ряда.

Погрешность вычисления значения функции y при некотором значении x зависит от количества членов ряда и значения x.

Поэтому расчет заканчивается при $|a_k| < eps$, где eps – допустимая погрешность расчетов.

© С.В.Кухта, 2009

Пример 3

```
PROGRAM SIN_R;
Var  y, S, x, eps, a, p: real; k: Word;
Begin
  Write ('Введите значение аргумента x=');
  readln(x);
  Write ('Введите значение погрешности eps=');
  readln(eps);
  Writeln;
```

© С.В.Кухта, 2009

Пример 3

```
  y:= sin(x);
  k:= 1;      a:= x;    { a  - первый член ряда }
  S:= a;                { S - первая частичная сумма ряда }
  While   abs(a) >= eps  do   begin
        k:= k+2;
        p:= -x*x/(k*(k-1));
        a:= a*p;
        S:= S+a;
         end;
  Writeln('Приближ. значение ф-ции =', S:7:3);
  Writeln('Контр. значение ф-ции sin(x)=',
         y:7:3);
End.
```

© С.В.Кухта, 2009

Цикл с постусловием

Оператор используется, когда количество повторений заранее неизвестно, а задано некоторое условие выхода из цикла.

Формат:

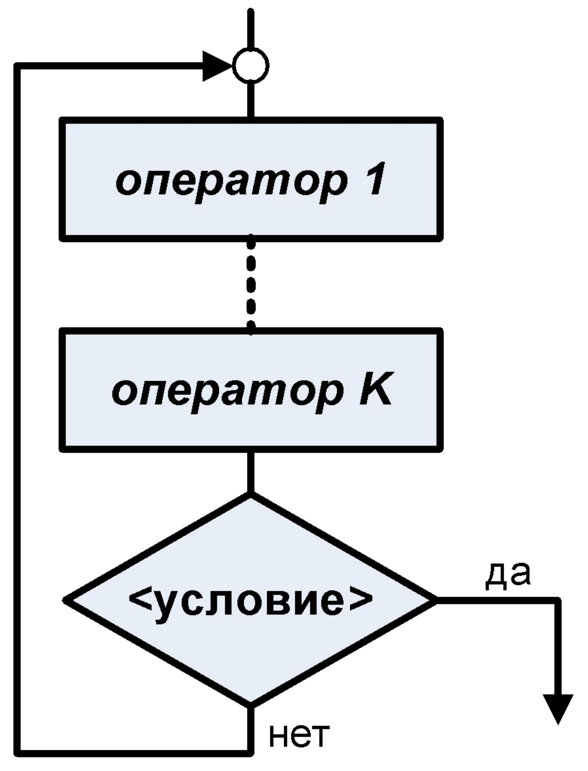
```
Repeat
        <оператор 1>;
        ...
       <оператор K>
Until  <условие>;
```

Цикл с постусловием – это цикл, в котором проверка условия выполняется в конце цикла.

© С.В.Кухта, 2009

Цикл с постусловием

Схема выполнения оператора:



Последовательность действий при выполнении цикла:

1. Выполняются ***<оператор 1>***, … ***<оператор K>***.
2. Проверяется ***<условие>***. Если оно ложно, то выполняется возврат к метке ***Repeat***, т.е. к выполнению операторов тела цикла.
3. Пункты 1 и 2 выполняются до тех пор, пока ***<условие>*** не станет истинным.

© С.В.Кухта, 2009

! Обратите внимание, что на каждой итерации циклы ***for*** и ***while*** выполняют только по одному оператору (либо группу операторов, заключенную в операторные скобки ***begin-end*** и потому воспринимаемую как единый составной оператор).

В отличие от них, цикл ***repeat-until*** позволяет выполнить сразу несколько операторов: ключевые слова ***repeat*** и ***until*** сами служат операторными скобками.

© С.В.Кухта, 2009

Пример 1: цикл с постусловием

Задача: Ввести целое **положительное** число (<2000000) и определить число цифр в нем.

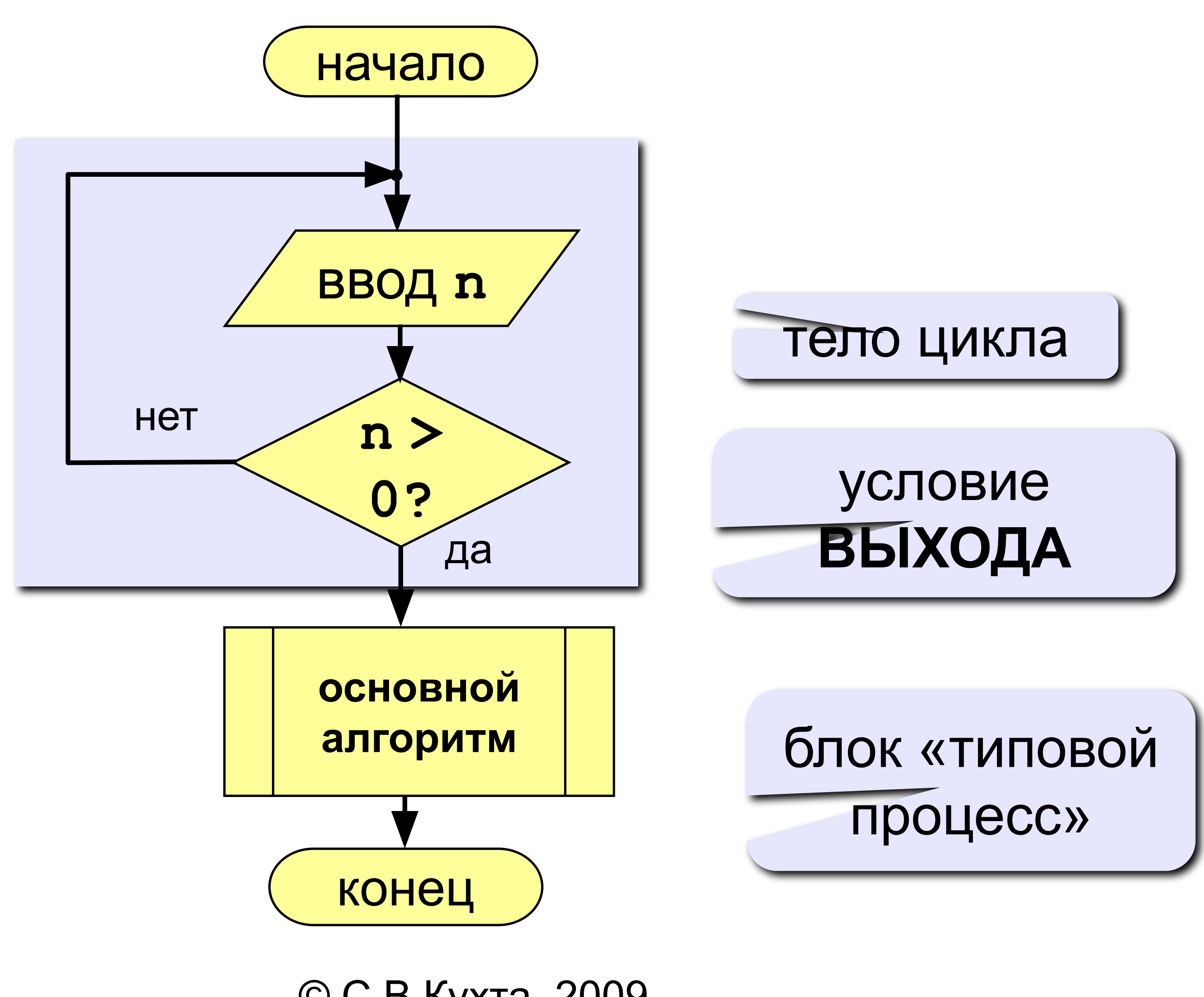
Проблема: Как не дать ввести отрицательное число или ноль?

Решение: Если вводится неверное число, вернуться назад к вводу данных (цикл!).

Особенность: Один раз тело цикла надо сделать в любом случае => проверку условия цикла надо делать в конце цикла (цикл с **постусловием**).

© С.В.Кухта, 2009

Цикл с постусловием: алгоритм



© С.В.Кухта, 2009

Программа

```
program qq;
var n: integer;
begin
  repeat
    writeln('Введите положительное число');
    read(n);
  until n > 0;
  ... { основной алгоритм }
end.
```

условие **ВЫХОДА**

Особенности:

- тело цикла всегда выполняется хотя бы один раз
- после слова **until** ("до тех пор, пока не…") ставится условие **ВЫХОДА** из цикла

© С.В.Кухта, 2009

Сколько раз выполняется цикл?

```
a := 4; b := 6;
repeat a := a + 1; until a > b;
```

3 раза
a = 7

```
a := 4; b := 6;
repeat a := a + b; until a > b;
```

1 раз
a = 10

```
a := 4; b := 6;
repeat a := a + b; until a < b;
```

зацикливание

```
a := 4; b := 6;
repeat b := a - b; until a < b;
```

2 раза
b = 6

```
a := 4; b := 6;
repeat a := a + 2; until a < b;
```

зацикливание

© С.В.Кухта, 2009

Пример 1

Задача: решить уравнение

$$x^2 = 5\cos x \quad \Leftrightarrow \quad x^2 - 5\cos x = 0$$

$$f(x) = 0$$

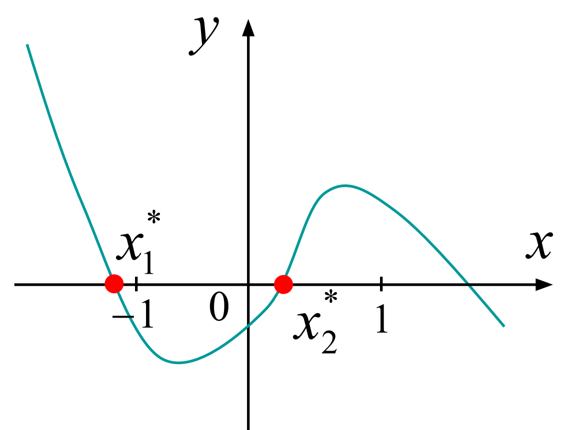
Типы решения:

- **аналитическое** (точное, в виде формулы)

$$x^* = \ldots$$

- **приближенное** (неточное)

графический метод



численные методы

$x_0 = -1$ **начальное приближение**

$x_1 = -1{,}102$

$x_2 = -1{,}215$

$$x^* \approx -1{,}252\ldots$$

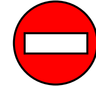
© С.В.Кухта, 2009

Численные методы

Идея: последовательное уточнение решения с помощью некоторого алгоритма.

Область применения: когда найти точное решение невозможно или крайне сложно.

1) можно найти хоть какое-то решение
2) во многих случаях можно оценить ошибку (то есть можно найти решение **с заданной точностью**)



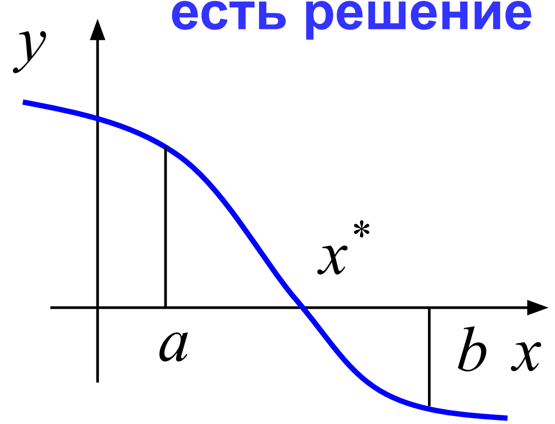
1) нельзя найти *точное* решение

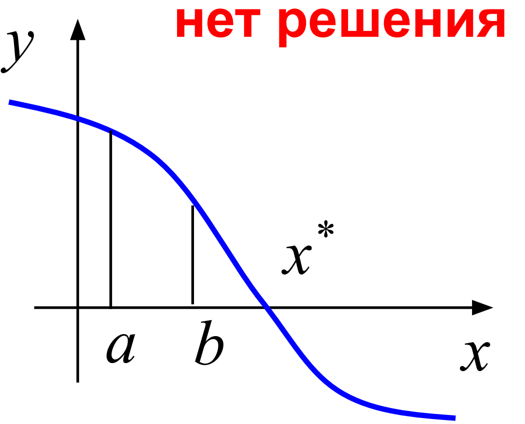
$$\sqrt{x+1} - 4\sin(x-1) = 0 \quad \Rightarrow \quad x = 1{,}3974 \quad x \approx 1{,}3974$$

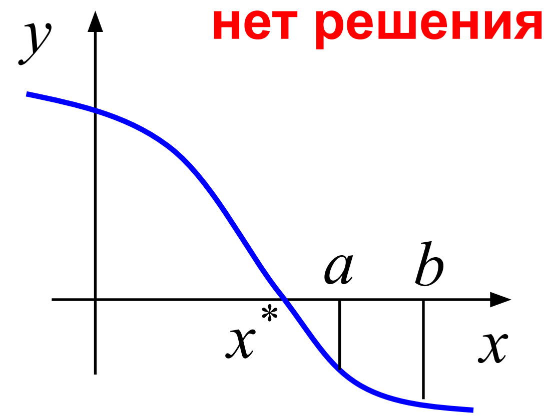
2) невозможно исследовать решение при изменении параметров
3) большой объем вычислений
4) иногда сложно оценить ошибку
5) нет универсальных методов

© С.В.Кухта, 2009

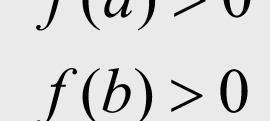
Есть ли решение на [*a*, *b*]?







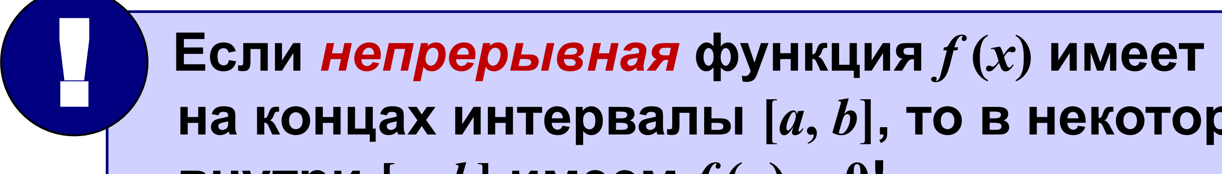
$$f(a) > 0$$
$$f(b) < 0$$



$$f(a) > 0 \qquad f(a) < 0$$
$$f(b) > 0 \qquad f(b) < 0$$

$$f(a)f(b) < 0$$

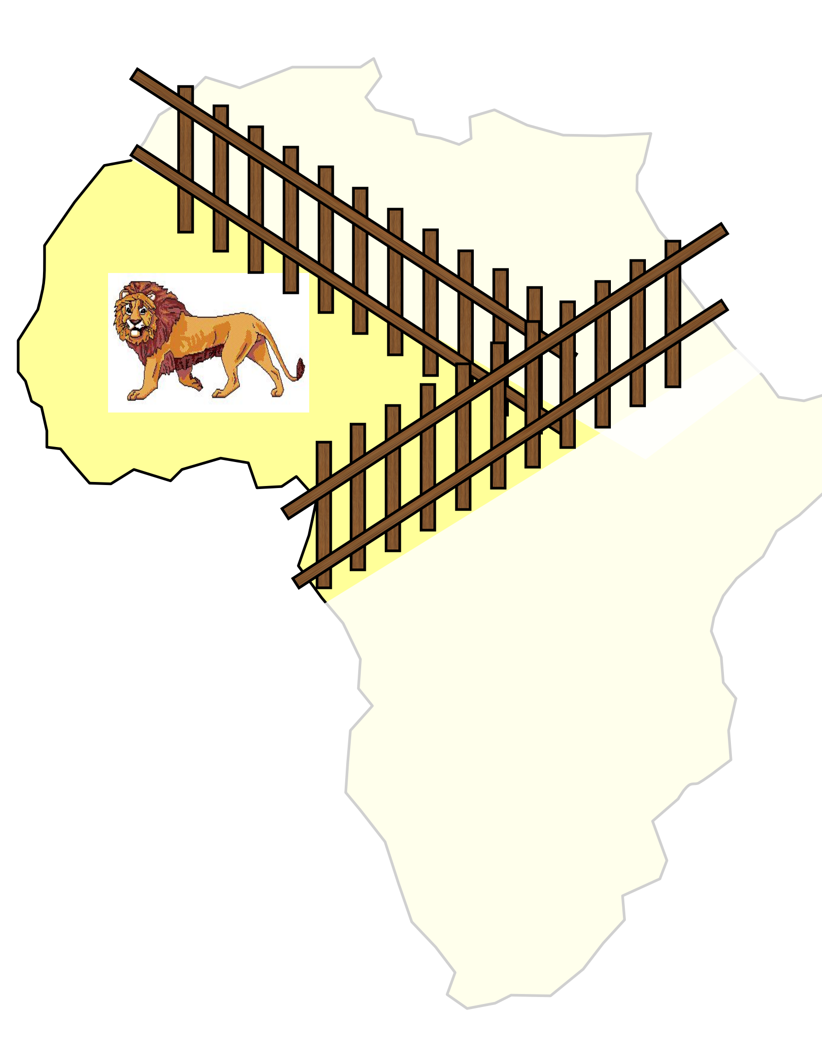
$$f(a)f(b) > 0$$

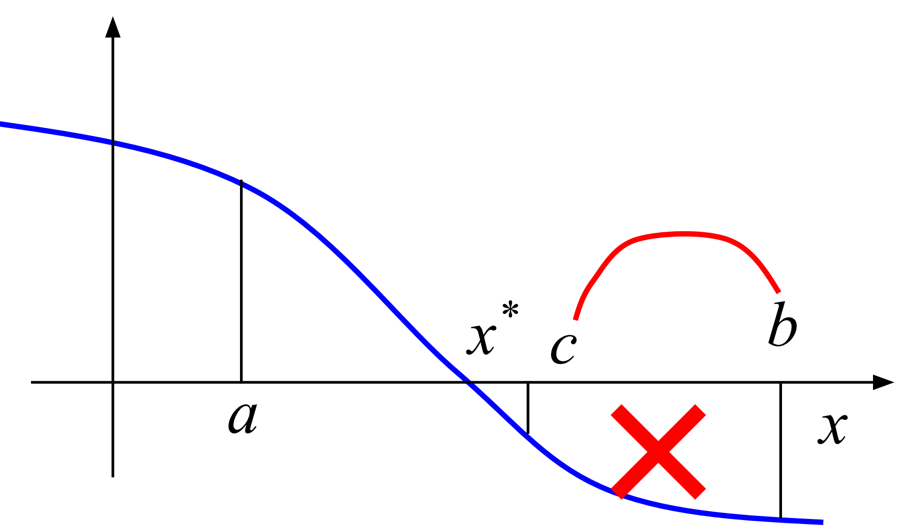


Если *непрерывная* функция $f(x)$ имеет разные знаки на концах интервалы $[a, b]$, то в некоторой точке внутри $[a, b]$ имеем $f(x) = 0$!

© С.В.Кухта, 2009

Метод дихотомии (деление пополам)

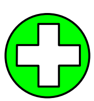




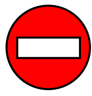
1. Найти середину отрезка `[a,b]`:
   `c = (a + b) / 2;`
2. Если `f(c)*f(a)`<0, сдвинуть правую границу интервала
   `b = c;`
3. Если `f(c)*f(a)`≥ 0, сдвинуть левую границу интервала
   `a = c;`
4. Повторять шаги 1-3, пока не будет `b – a` ≤ `ε`.

© С.В.Кухта,

Метод дихотомии (деления пополам)



- **простота**
- можно получить решение с заданной **точностью** (в пределах точности машинных вычислений)



- нужно знать **интервал** $[a, b]$
- на интервале $[a, b]$ должно быть только **одно** решение
- **большое число шагов** для достижения высокой точности
- только для функций **одной** переменной

© С.В.Кухта, 2009

Метод деления отрезка пополам

```
Program  TransEquation;
 Const   a = 0;   b = 1;  Eps = 1e-5;
 Var u, v, Root : Real;  Fu, Fr : Real;
 Begin
      u:=a;  v:=b;       { инициализация цикла }
      Fu := a*a-5*Cos(a);
    Repeat
      Root :=(u+v)/2; { средняя точка отрезка }
      Fr :=Root*Root-5*Cos(Root);
      if  Fu*Fr > 0    { выбор правой или левой половины
      }
         then  begin
        u:=Root; Fu:=Fr  end
         else   v:=Root
    until Abs(v-u)<Eps;
    writeln (' корень= ', Root)
 End
```

5. Операторы ограничения и прерывания цикла

© С.В.Кухта, 2009

Оператор *break*

Существует возможность прервать выполнение цикла (или одной его итерации), не дождавшись конца его (или ее) работы.

Оператор ***break*** прерывает работу всего цикла и передает управление на следующий за ним оператор, при этом не контролируется условие выхода из цикла.

Формат:

```
Break;
```

При прерывании работы циклов ***for-to*** и ***for-downto*** с помощью процедуры ***break*** переменная цикла (счетчик) сохраняет свое текущее значение, не "портится".

© С.В.Кухта, 2009

Оператор *continue*

Действие оператора **Continue** заключается в передаче управления на начало цикла, при этом контролируется условие выхода из цикла, т.е. прерывается работа текущей итерации цикла и передается управление:

- следующей итерации (цикл ***repeat-until***)
- или на предшествующую ей проверку (циклы ***for-to***, ***for-downto***, ***while***).

Формат:

```
Continue;
```

© С.В.Кухта, 2009

Пример

Пример использования операторов для блокировки несанкционированного доступа в программу

```
For  i:= 1  to  3   do   begin
     Write( 'Введите ПАРОЛЬ:' );
     Readln(S);   {S и Parol - переменные одного типа}
     If  S=Parol Then Break   { прерывание цикла }
     else   If  i<>3 Then Continue;
        { ограничение цикла }
     Writeln( 'Доступ к программе ЗАПРЕЩЕН' );
     Writeln( 'Нажмите Enter' );
     Readln;
     Halt              { прерывание программы }
end;
```

© С.В.Кухта, 2009

Оператор *goto*

Возвращаясь к сказанному об операторе ***goto***, необходимо отметить, что при всей его нежелательности все-таки существует ситуация, когда предпочтительно использовать именно этот оператор - как с точки зрения структурированности текста программы, так и с точки зрения логики ее построения, и уж тем более с точки зрения уменьшения трудозатрат программиста.

Эта ситуация - *необходимость передачи управления изнутри нескольких вложенных циклов на самый верхний уровень*.

© С.В.Кухта, 2009

Оператор *goto*

Дело в том, что процедуры ***break*** и ***continue*** прерывают только один цикл - тот, в теле которого они содержатся.

Поэтому в упомянутой выше ситуации пришлось бы заметно усложнить текст программы, вводя много дополнительных прерываний.

А один оператор ***goto*** способен заменить их все.

Сравним, например, два программно-эквивалентных отрывка.

© С.В.Кухта, 2009

Пример

```
write('Матрица ');
for i:=1 to n do begin
 flag:=false;
 for j:=1 to m do
  if a[i,j]>a[i,i] then
     begin
    flag:=true;
    write('не ');
    break;
     end
 if flag then break;
end;
writeln('обладает свойством
  диагонального
  преобладания');
```

```
write('Матрица ');
for i:=1 to n do

for j:=1 to m do
  if a[i,j]>a[i,i] then
     begin

    write('не ');
    goto 1;
     end

1:
writeln('обладает свойством
  диагонального
  преобладания');
```

© С.В.Кухта, 2009

6. Поиск элемента в массиве

© С.В.Кухта, 2009

Постановка задачи

Пусть **A = {a_1, a_2, ...}** – последовательность однотипных элементов и ***b*** – некоторый элемент, обладающий свойством ***P***. **Найти место элемента *b*** в последовательности ***A***.

© С.В.Кухта, 2009

Постановка задачи

Поскольку представление последовательности в памяти может быть осуществлено в виде массива, задачи могут быть уточнены как одна из следующих задач поиска элемента в массиве *A*:

- найти максимальный (минимальный) элемент массива;
- найти заданный элемент массива;
- найти *k*–ый по величине элемент массива.

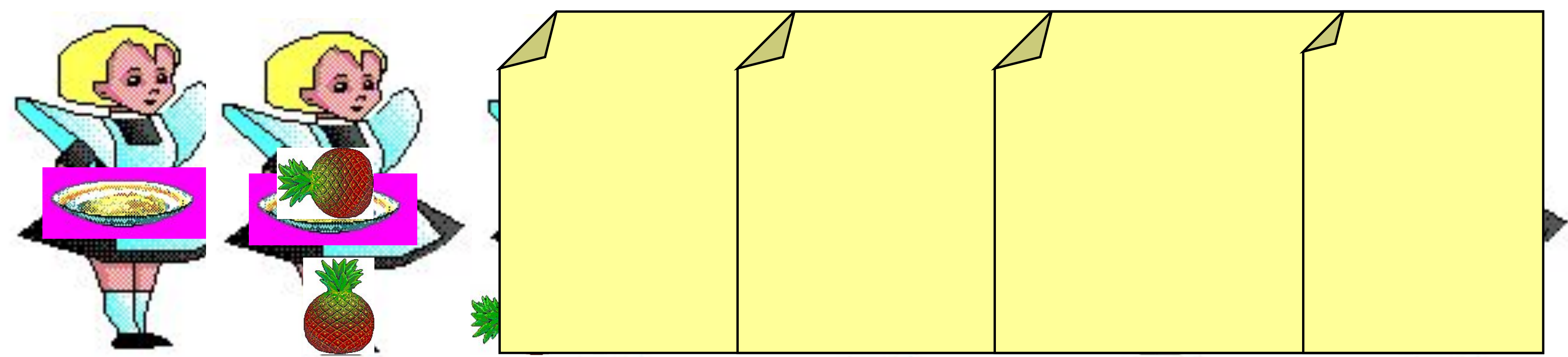
Наиболее простые и часто оптимальные алгоритмы основаны на последовательном просмотре массива ***A*** с проверкой свойства ***P*** на каждом элементе.

© С.В.Кухта, 2009

Максимальный элемент

Задача: найти в массиве максимальный элемент.

Алгоритм:



Псевдокод:

```
{ считаем, что первый элемент – максимальный }
for i:=2 to N do
  if a[i] > { максимального } then
    { запомнить новый максимальный элемент a[i] }
```

© С.В.Кухта, 20

? Почему цикл от i=2?

Максимальный элемент

Дополнение: как найти номер максимального элемента?

```
                   { считаем, что первый – максимальный }
iMax := 1;
for i:=2 to N do        { проверяем все остальные }
  if a[i] > a[iMax] then { нашли новый максимальный }
  begin
                          { запомнить a[i] }
    iMax := i;            { запомнить i }
  end;
```

Как упростить?

По номеру элемента iMax всегда можно найти его значение a[iMax]. Поэтому везде меняем max на a[iMax] и убираем переменную max.

© С.В.Кухта, 2009

Максимальный элемент

```
program qq;
const N = 5;
var a: array [1..N] of integer;
    i, iMax: integer;
begin
  writeln('Исходный массив:');
  for i:=1 to N do begin
      a[i] := random(100) + 50;
      write(a[i]:4);
  end;
  iMax := 1; { считаем, что первый – максимальный }
  for i:=2 to N do      { проверяем все остальные }
    if a[i] > a[iMax] then { новый максимальный }
      iMax := i;            { запомнить i }
  writeln; {перейти на новую строку}
  writeln('Максимальный элемент a[', iMax, ']=', a[iMax]);
end.
```

случайные числа в интервале [50,150)

поиск максимального

Поиск заданного элемента в массиве

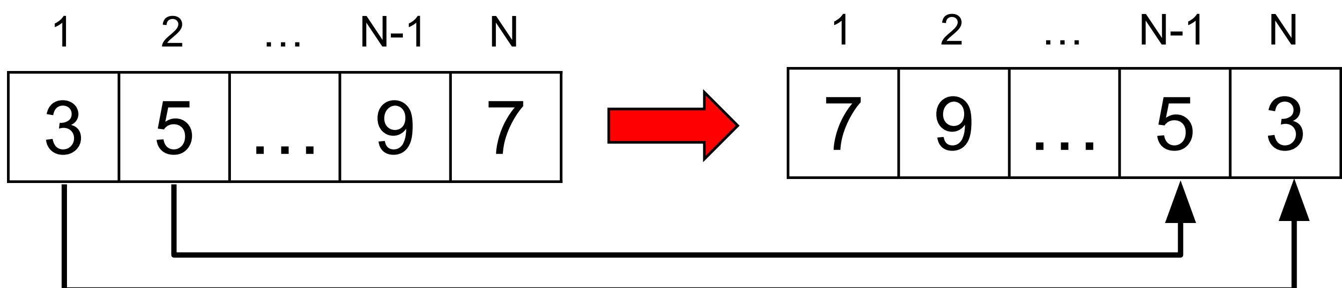
```
Program Search_in_Array;
 Label   1;
 Const   n = 100;
 Var   A : Array[1..n]  of  Real;
     b : Real;   Flag : Boolean;   i : Integer;
 Begin
       Writeln('Введите массив'); {Блок ввода массива}
       For  i:=1 to n do   Read (A[i]);
       Writeln('Введите элемент для поиска');
       Read (b);
       Flag := true;
       For  i:=1 to n  do
         If  A[i] = b then  begin  { прерывание цикла }
    Flag := false;   goto  1   end;
    1: If  Flag  then Writeln('Элемента ', b,
               ' в массиве нет')
    else Writeln('Элемент ', b, ' стоит на ',
              i, '-м месте');
 End.
```

7. Обработка массивов

© С.В.Кухта, 2009

Реверс массива

Задача: переставить элементы массива в обратном порядке.



Алгоритм:

сумма индексов N+1

поменять местами **A[1]** и **A[N]**, **A[2]** и **A[N-1]**, …

Псевдокод:

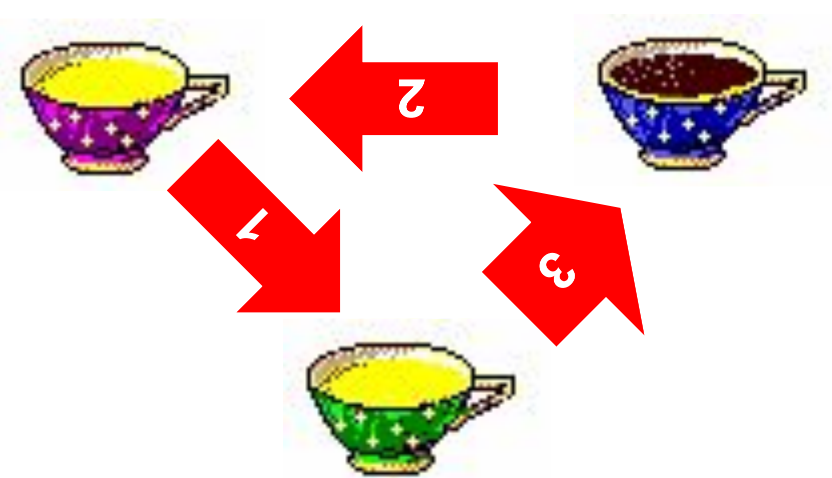
```
for i:=1 to N div 2 do
  { поменять местами A[i] и A[N+1-i] }
```

Что неверно?

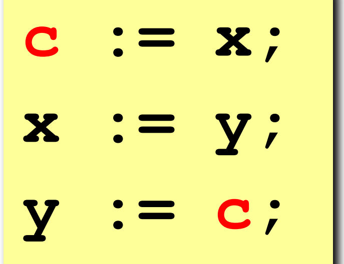
© С.В.Кухта, 2009

Как переставить элементы?

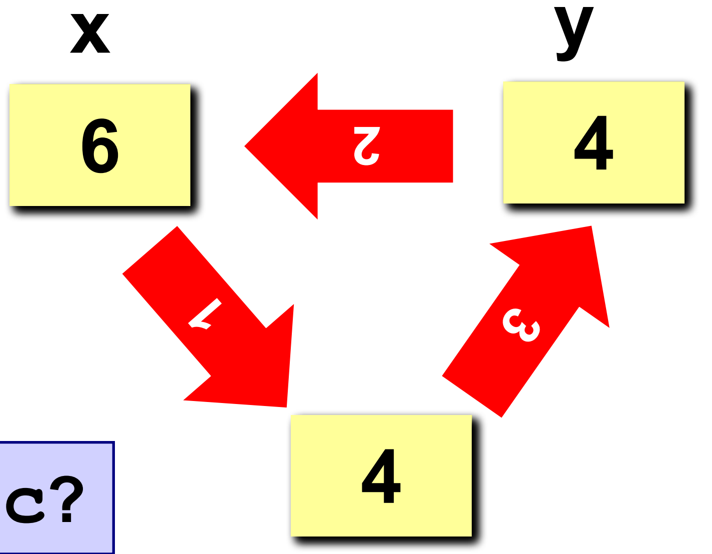
Задача: поменять местами содержимое двух чашек.

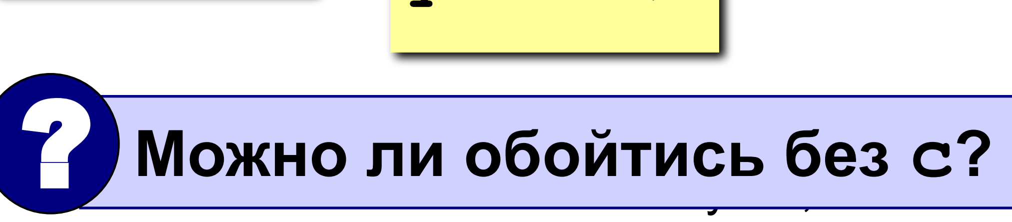


Задача: поменять местами содержимое двух ячеек памяти.




```
c := x;
x := y;
y := c;
```





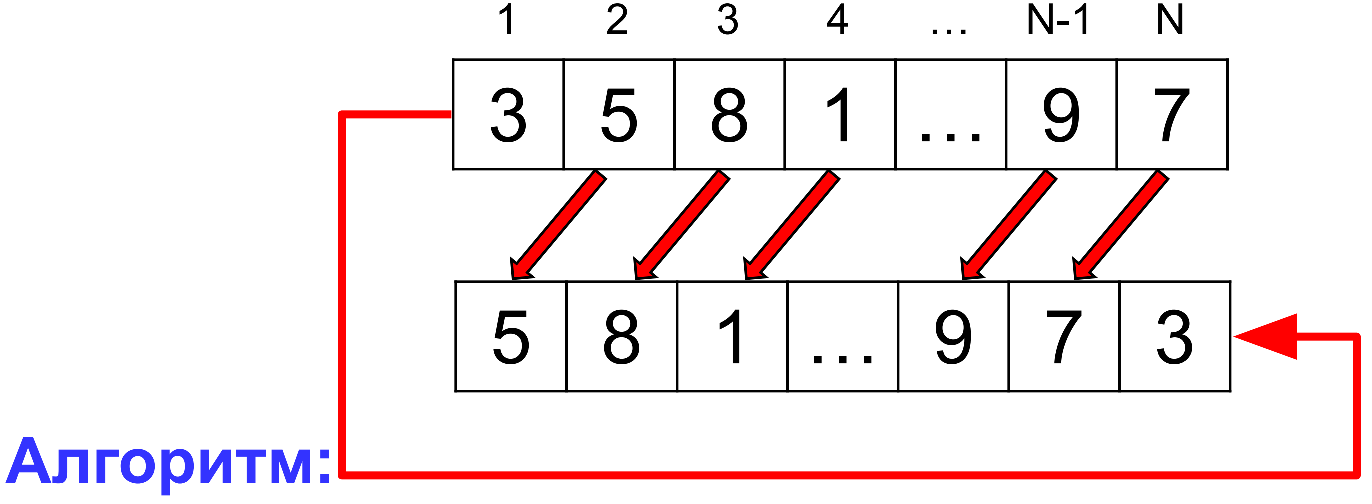
Программа

```
program qq;
const N = 10;
var A: array[1..N] of integer;
    i, c: integer;
begin
  { заполнить массив }
  { вывести исходный массив }
  for i:=1 to N div 2 do begin
     c:=A[i]; A[i]:=A[N+1-i]; A[N+1-i]:=c;
   end;
  { вывести полученный массив }
end.
```

© С.В.Кухта, 2009

Циклический сдвиг

Задача: сдвинуть элементы массива влево на 1 ячейку, первый элемент становится на место последнего.



Алгоритм:

```
A[1]:=A[2]; A[2]:=A[3];… A[N-1]:=A[N];
```

Цикл:

почему не **N**?

```
for i:=1 to N-1 do
  A[i]:=A[i+1];
```

Что неверно?

Программа

```
program qq;
const N = 10;
var A: array[1..N] of integer;
    i, c: integer;
begin
  { заполнить массив }
  { вывести исходный массив }
   c := A[1];
   for i:=1 to N-1 do A[i]:=A[i+1];
   A[N] := c;
  { вывести полученный массив }
end.
```

© С.В.Кухта, 2009

8. Способы перебора элементов массивов

© С.В.Кухта, 2009

Часто при работе с массивами задача ставится так, что требуется все элементы или их часть обработать одинаково. Для такой обработки организуется ***перебор элементов***.

Схему перебора элементов массива можно охарактеризовать:

- направлением перебора;
- количеством одновременно обрабатываемых элементов;
- характером изменения индекса.

По ***направлению перебора*** различают схемы:

- от первого элемента к последнему (от начала массива к концу);
- от последнего элемента к первому (от конца к началу);
- от обоих концов к середине.

© С.В.Кухта, 2009

В массиве одновременно можно обрабатывать один, два, три и т.д. элемента.

Часто в качестве параметра цикла используется индекс массива.

Обратите внимание также на то обстоятельство, что после изменения индекса его необходимо сразу же проверить на попадание в заданный диапазон, иначе возможны ошибки.

© С.В.Кухта, 2009

Общие правила организации перебора

В правильно построенной схеме обязательно должны присутствовать:

- блок установки начальных значений индексов массива,
- блок проверки индекса (индекс не должен выходить за границы индексов массива),
- блок изменения индекса для перехода к следующему элементу массива, причем, за блоком изменения индекса по времени выполнения должен располагаться блок проверки индекса на принадлежность интервалу, определенному границами массива.

Если будет нарушено хотя бы одно из перечисленных условий, то в процессе выполнения программы возникнут ошибки

© С.В.Кухта, 2009

Случай 1

Перебрать элементы массива по одному, двигаясь от начала массива к концу.

Здесь индекс начального элемента 1, индекс последнего обрабатываемого элемента n, шаг перебора 1. Конечное значение (***кз***) параметра цикла при условии проверки окончания с помощью сравнения **<=** может быть вычислено по формуле:

(конечное значение - начальное значение + 1)/шаг = n.

Отсюда: ***(кз-1+1)/1=n*** или ***кз=n***.

© С.В.Кухта, 2009

Случай 1

Схема перебора может быть представлена в виде:

```
for i:=1 to n do
    { обработка a[i] }
```

или:

```
i:=1;
while i<=n do begin
   { обработка a[i] }
  i:=i+1
end;
```

© С.В.Кухта, 2009

Случай 1

Если условие окончания проверяется с помощью сравнения **<**, то конечное значение вычисляется так:

(конечное значение - начальное значение)/шаг=n.

Отсюда: ***(кз - 1)/1=n*** или ***кз=n+1***.

Схема перебора может быть представлена в виде:

```
i:=1;
while  i<n+1  do  begin
     { обработка a[i] }
   i:=i+1
end;
```

© С.В.Кухта, 2009

Случай 2

Перебрать элементы массива по одному, двигаясь от конца массива к началу.

```
for i:=n downto 1 do
    { обработка a[i] }
```

или:

```
i:=n;
while i>=1 do begin
   { обработка a[i] }
  i:=i-1
end;
```

© С.В.Кухта, 2009

Случай 3

Обработать массив по одному элементу, двигаясь с обоих концов к середине массива.

```
i:=1;  {установка нижней границы}
j:=n;  {установка верхней границы}
while  i<=j  do  begin
        { обработать элемент a[i] }
   i:=i+1;
        { обработать элемент a[j] }
   j:=j-1;
End;
```

© С.В.Кухта, 2009

Случай 4

Перебрать элементы массива с четными индексами, двигаясь от начала к концу.

Вариант 1. Здесь индекс начинает изменяться с четного числа, величина шага, равная двум, обеспечивает сохранение четности индекса.

```
i:=2;
while  i<=n  do begin
     { обработка a[i] }
   i:=i+2
end;
```

Вариант 2. Здесь внутрь цикла перебора вложен оператор, проверяющий четность индекса (работает медленнее).

```
for  i:=1  to  n  do
   if   i mod 2=0  then
    { обработка a[i] };
```

Случай 5

Перебрать элементы массива с четными индексами, двигаясь от конца массива к началу.

Вариант 1. Здесь установка начального значения - не простое присваивание, а условный оператор, позволяющий отыскать последний элемент массива с четным индексом.

```
if  n mod 2 =0 then
      i:=n
else  i:=n-1;
while  i>0  do  begin
    { обработка a[i] };
  i:=i-2
end;
```

© С.В.Кухта, 2009

Случай 5

<u>Вариант 2.</u> Условный оператор, устанавливающий начальное значение индекса, можно внести в тело цикла.

```
i:=n;
while  i>0  do  begin
   if  i mod 2 =0  then
      { обработка a[i] }  ;
   i:=i-1
end;
```

© С.В.Кухта, 2009

Случай 6

Перебрать элементы массива с четным индексом, двигаясь с обоих концов массива к его середине.

Для решения этой задачи соединим схемы перебора, рассмотренные в случаях 4 и 5:

```
i:=2;
if  n mod 2 =0  then j:=n
else  j:=n-1;
while  i<=j  do begin
    { обработать элемент a[i] };
  i:=i+2;
    { обработать элемент a[j] };
  j:=j-2;
end;
```

© С.В.Кухта, 2009

Случай 7

Перебрать элементы массива с индексом, кратным k, двигаясь от начала массива к его концу.

Для решения этой задачи соединим схемы перебора, рассмотренные в случаях 4 и 5:

```
i:=k;
while  i<=n do begin
    {обработка a[i] }
  i:=i+k
end;
```

© С.В.Кухта, 2009

Случай 8

Перебрать соседние элементы массива, двигаясь от начала массива к концу (случай двух соседей).

Для массива из 5 элементов нужно последовательно обработать пары:

a[1] и ***a[2]***, ***a[2]*** и ***a[3]***, ***a[3]*** и ***a[4]***, ***a[4]*** и ***a[5]***.

Вариант 1.

```
for  i:=1  to  n-1  do
   { обработать a[i] и a[i+1] };
```

Вариант 2.

```
for  i:=1  to  n-1  do
   { обработать a[i-1] и a[i] };
```

© С.В.Кухта, 2009

Случай 8

Перебрать соседние элементы массива, двигаясь от начала массива к концу (случай трех соседей).

Для массива из 5 элементов нужно последовательно обработать пары:

a[1]-***a[2]***-***a[3]***, ***a[2]***-***a[3]***-***a[4]***, ***a[3]***-***a[4]***-***a[5]***.

Вариант 1.

```
for i:=1 to n-2  do
   { обработать a[i] - a[i+1] - a[i+2] };
```

Вариант 2.

```
for i:=2 to n-1  do
   { обработать a[i-1] - a[i] - a[i+1] };
```

Вариант 3.

```
for i:=3 to n  do
   { обработать a[i-2] - a[i-1] - a[i] };
```

© С.В.Кухта, 2009

8. Примеры решения некоторых типовых задач

© С.В.Кухта, 2009

Задача 1

Последовательность элементов задана формулой общего члена $a_i = sin(i+i/n)$, где i изменяется от 1 до n.

Написать программу для нахождения первого элемента последовательности, большего заданного числа Z.

© С.В.Кухта, 2009

Задача 1: решение

Исходными данными для решения задачи являются элементы последовательности:

a_1=sin(1+1/n), a_2=sin(2+2/n), ..., a_n=sin(n+n/n).

В результате получаем либо номер элемента, большего заданного **Z**, либо ответ: «Такого элемента нет».

Решение этой задачи может закончиться по двум причинам:

1) перебрали все элементы последовательности и не нашли нужного элемента;
2) в процессе перебора обнаружился элемент, больший **Z**. В этом случае перебор прекращается и формируется ответ.

Первую причину окончания можно определить, проверив условие: ***i* > *n***, где ***i*** - текущий элемент последовательности, а ***n*** - количество элементов в ней.

© С.В.Кухта, 2009

Задача 1: решение

Вторая причина имеет два значения:

- ✔ «найдено»
- ✔ или «не найдено».

Поэтому ее можно изображать логическим значением, где ***true*** соответствует найдено, а ***false*** - не найдено.

Таким образом, условие окончания поиска может быть записано в виде

(i>n) or f,

что соответствует фразе русского языка: «Просмотрены все элементы или найдено».

Просмотр элементов последовательности в цикле будет выполняться в случае ложности приведенного условия.

© С.В.Кухта, 2009

Задача 1: программа

```
program  pr4;
var   n, i : integer;
   z: real;       { заданное число }
     f: boolean;   { true, если найден искомый элемент }
Begin
   write('Введите n и z ');
   readln(n, z);
   i:=1;       f:=false;
   while  (i<=n) and  not  f  do
     if  sin(i+i/n)>z  then
      f:=true      { искомый элемент найден }
     else   i:=i+1;     { переходим к след. элементу }
   if  f then
   writeln('Номер 1-го элемента больше ', z,
         ' =', i)
   else writeln('Нет элементов больших ', z);
end.
```

Задача 2

Вычислить значение выражения:

y=cos(1+cos(2+cos(3+...+cos(n-1+cos(n)...)).

Задача 2: решение

Для начала запишем заданную формулу при различных значениях ***n***.

Для *n=3* получим *y=cos(1+cos(2+cos(3)))*.

Для *n=5* получим *y=cos(1+cos(2+cos(3+cos(4+cos(5)))))*.

Чтобы вычислить это выражение без компьютера, «вручную», нужно начать вычисления с самых вложенных скобок:

1. *y1=cos(5)*
2. *y2=cos(4+cos(5))* или с учетом первого шага *y2=cos(4+y1)*
3. *y3=cos(3+y2)*
4. *y4=cos(2+y3)*
5. *y5=cos(1+y4)*

Таким образом, в теле цикла будет повторяться оператор

y=cos(i+y),

где *i* изменяется от *n* до 1.

© С.В.Кухта, 2009

Задача 2: программа

Фрагмент программы решения задачи:

```
y:=0;
for i:=n downto 1 do
   y:=cos(i+y);
```

© С.В.Кухта, 2009

Задача 3

Вычислить для произвольного натурального n

$$y = \sqrt{n + \sqrt{n - 1 + ... + \sqrt{2 + \sqrt{1}}}}$$

Решение.

Этот пример очень похож на предыдущий, однако здесь вместо функции ***cos(x)*** используется квадратный корень и натуральные числа под корнями уменьшаются.

Отсюда получаем фрагмент программы:

```
y:=0;
for  i:=1  to  n  do
     y:= sqrt(i+y);
```

© С.В.Кухта, 2009

Задача 4

Вычислить для произвольного натурального n

$$1+\cfrac{x^n}{2+\cfrac{x^{n-1}}{3+\cfrac{x^{n-2}}{4+\cfrac{\cdots}{n+x}}}}$$

Решение. Этот пример также похож на предыдущие. В нем используются функция деления и натуральные числа от **1** до ***n***. Аналогично предыдущим примерам, здесь повторяется следующий оператор: ***y=i+p/y***, где ***p*** - степень ***x***.

Степень можно получить последовательными умножениями на ***x***.

```
y:=1;    p:=1;    { степени x }
for i:=n downto  1 do begin
  p:=p*x;
  y:=i+p/y
end;
```

Задача 5

Дано натуральное число n. Поменять порядок цифр числа n на обратный.

Решение. Например, для ***n*=1829** должно получиться ***m*=9281**. Для определения последней цифры числа нужно найти остаток от деления этого числа на 10, а для отбрасывания этой цифры - найти частное от деления целых чисел. Но последовательно получаемые цифры числа не отбрасываются, а результат ***m*** домножается на **10** и цифра прибавляется к результату.

Цикл выполняется до тех пор, пока есть цифры в исходном числе.

```
s:=0;      { сумма цифр числа n }
while  n>0  do  begin { пока есть цифры в числе n }
   s:=s+n mod 10; { прибавить последнюю цифру к сумме }
   n:=n div 10      { отбросить последнюю цифру }
end;
```

© С.В.Кухта, 2009

Задача 6

Последовательность элементов задана формулой общего члена a_i=*sin(i+i/n)*, где *i* изменяется от 1 до *n* (*n* - натуральное). Найти максимальный элемент и его номер в последовательности.

Решение.

Первый элемент последовательности объявляем кандидатом на максимум.

Последовательно сравниваем все остальные элементы последовательности с кандидатом на максимум:

- если очередной элемент меньше или равен кандидату, то переходим к следующему элементу;
- если очередной элемент больше кандидата, то заменяем им кандидата и продолжаем сравнения.

© С.В.Кухта, 2009

Задача 6: программа

```
program   primer;
var   n, i : integer;
  a: real;        { элемент посл-сти }
  max: real;          { максимальный элемент }
  mi: integer;      { номер максимального элемента }
begin
  write('введите число ');
  readln(n);
  max:=sin(1+n/1);    mi:=1;
  i:=2;
  while i<=n do begin
    a:=sin(i+n/i);
  if  max<a  then   begin
   max:=a;  mi:=i end;
  i:=i+1;
  end;
  write('max=', max:15:2, ' номер max=', mi);
end.
```

Задача 7

Баржа, двигаясь по реке, ежедневно проходит расстояние, определяемое формулой

$$y = \begin{cases} y_{i-1} + 0.5 \cdot y_{i-2}, & \text{если i - четное} \\ 0.7 \cdot y_{i-5} - 0.2 \cdot y_{i-4}, & \text{если i - нечетное} \end{cases}$$

Определить, сможет ли баржа за *n* дней пройти расстояние в *z* километров, если в каждый из первых пяти дней она проходила по одному километру.

Решение.

Чтобы вычислить дневной путь баржи, начиная с шестого дня, необходимо знать расстояния, пройденные баржей за предыдущие пять дней.

Введем обозначения: y_i - ***a***, y_{i-1} - ***b***, y_{i-2} - ***c***, y_{i-3} - ***d***, y_{i-4} - ***e***, y_{i-5} - ***f***.

Необходимо найти путь баржи за *n* дней (сумма расстояний) и сравнить его с z.

© С.В.Кухта, 2009

Задача 7: программа

```
Var  a, b, c, d, e, f: real;
     i, n: integer;   { i - счетчик, n - количество дней }
     s: real; { сумма расстояний, пройденных за n дней }
     z: real; { расстояние для сравнения }
begin
  f:=1;  e:=1;  d:=1;  c:=1;  b:=1;  s:=5;
  write('Введите количество дней ');
     readln(n);
  for i:=5 to n do begin
     if i mod 2=0 then  a:=b+0.5*c
  else  a:=0.7*f-0.2*e;
  f:=e;  e:=d; d:=c; c:=b; b:=a;
  s:=s+a;
  end;
 writeln('Пройденное расстояние ', s)
 if  s>z then write(' больше ', z)
 else  if s=z then write(' равно ', z)
   else  write(' меньше ', z)
end.
```

Задача 8

Последовательность (a_n) задается так: a_1 - некоторое натуральное число, a_{n+1} - сумма квадратов цифр числа a_n, n>=1. Найти m-й член последовательности.

Решение.

Для решения задачи нужно ***m*** раз выполнить вычисление очередного члена последовательности.

Для его вычисления необходим еще один цикл, в котором цифры последовательно отделяются от очередного члена последовательности и находится сумма их квадратов.

© С.В.Кухта, 2009

Задача 8: программа

```
   var a,   { член последовательности }
            s: integer; { сумма квадратов цифр предыдущего
                          члена посл-сти }
     i, m: integer;
begin
  write('Введите 1-й член посл-сти и m');
  readln(a, m);
  for i:=2 to m do begin
            s:=0;   { находим сумму квадратов цифр }
            While  a>0  do  begin
              s:=s + sqr(a mod 10);
              a:=a div 10
            end;
            a:=s    { запоминаем новый член последовательности }
  end;
end.
```

© С.В.Кухта, 2009

Задача 9

Возрастающая последовательность натуральных чисел вводится с клавиатуры до ввода числа 0. Вычислить и напечатать факториал каждого введенного числа.

Решение.

Здесь признаком окончания ввода является число **0**, заранее количество вводимых данных неизвестно, поэтому организуем итерационный цикл.

Исходные данные для этой задачи могут быть такими: **2 4 7 9 13 0**. Нужно получить: **2! 4! 7! 9! 13!**.

Заметим, что для вычисления факториала следующего числа можно использовать факториал предыдущего числа (по условию числа возрастают!).

Введем следующие обозначения: ***a*** - введенное число, ***b*** - число, с которого нужно продолжить вычисление факториала, ***i*** - счетчик выполненных умножений, ***p*** - факториал числа.

© С.В.Кухта, 2009

Задача 9: фрагмент программы

```
p:=1;  b:=1;
read(a);
while  a<>0  do  begin
      for  i:=b to a do
    p:=p*i;
   write('факториал ', a, ' = ', p, ' ');
   b:=a+1;
   read(a)
end;
```

© С.В.Кухта, 2009

Задача 10

Последовательность из n положительных чисел вводится с клавиатуры. Найти сумму максимального и минимального элементов последовательности: x_1, x_2+1/x_1, x_3+1/x_2, ... , x_{n+1}/x_{n-1}, где x_i - введенное число.

Решение.

Для ввода данных нужно организовать арифметический цикл по счетчику до ввода ***n*** чисел. Для работы на каждом шаге цикла нужно хранить два соседних числа.

Начальным значением для максимума и минимума может быть значение x_1.

© С.В.Кухта, 2009

Задача 10: фрагмент программы

```
read(a);     { вводим первое число посл-сти }
max:=a; min:=a;
for   i:=2   to   n   do  begin
     read(b);
  c:=b+1/a; { вычисляем элемент заданной посл-сти }
 { находим max и min }
  if  max<c  then   max:=c
  else   if  min>c  then  min:=c;
        a:=b   { запоминаем предыдущий элемент }
end;
write(max + min);
```

© С.В.Кухта, 2009

Задача 11

Для каждого четырехзначного числа составляется дробь: отношение суммы цифр числа к самому числу.
Найти четырехзначное число, для которого эта дробь наибольшая.

Решение.

Например, числу 1234 соответствует дробь 0,0081, а числу 9876 - 0,003.

Для перебора всех четырехзначных чисел организуем четыре вложенных цикла:

- в первом цикле будут перебираться цифры тысяч от 1 до 9 (ноль исключается, потому что число заведомо четырехзначное),
- во втором - сотни от 0 до 9,
- в третьем - десятки от 0 до 9,
- в четвертом цикле - единицы от 0 до 9.

Исходным значением искомого отношения может быть ноль.

© С.В.Кухта, 2009

Задача 11: фрагмент программы

```
y:=0;      { начальное значение отношения }
a:=0;      { искомое число }
for i:=1 to 9  do
  for j:=0 to 9  do
    for  k:=0 to 9  do
   for m:=0 to 9 do  begin
    s:=I + j + k + m;
    x:=((10 * I + j) * 10 + k) * 10 + m;
    if  y<s/x  then  begin
       y:=s/x;     a:=x   end
end;
```

© С.В.Кухта, 2009

Задача 12

Найти все натуральные трехзначные числа, каждое из которых обладает двумя следующими свойствами:

- первая цифра в три раза меньше последней его цифры;

- сумма самого числа с числом, получающимся из него перестановкой второй и третьей его цифр, делится на 8 без остатка.

Решение.

Простое решение можно получить по аналогии с предыдущим:

```
for i:=1 to 9 do        { цифра сотен }
  for j:=0 to 9 do      { цифра десятков }
 for k:=0 to 9 do begin { цифра единиц }
     c:=100*i+10*j+k;
   if (3*i=k) and ((c+100*i+10*k+j) mod 8=0)
     then write(c:4);
```

© С.В.Кухта, 2009

Задача 13

Любую целочисленную сумму денег большую 7 рублей можно выплатить без сдачи трешками и пятерками.

По заданному целому положительному *n* определить пару целых неотрицательных чисел *a* и *b*, таких, что $n=3 \cdot a+5 \cdot b$.

Решение. Значение количества выданных троек не может превысить ***n div 3***, а значение количества выданных пятерок не может превысить ***n div 5***.

Исходя из этого, организуем циклы перебора:

© С.В.Кухта, 2009

Задача 13: фрагмент программы

```
repeat
  write('Введите значение n>7 ');
  readln(n);
Until  n>7;
for a:=0 to n div 3 do
  for b:=0 to n div 5   do
  if n=3*a+5*b then
   writeln('3*', a, '+5*', b, '=', n);
```

© С.В.Кухта, 2009