Алгоритмические конструкции

Оглавление

Для учащихся Уинской СОШ

http://www.yaklass.ru/p/bosova-l-l-informatika-dlya-5-9-klassov/8-klass/osnovy-algoritmizatcii-14923/osnovnye-algoritmicheskie-konstruktcii-13987

Следование

Следование — алгоритмическая конструкция, отображающая естественный, последовательный порядок действий.

Алгоритмы, в которых используется только структура «следование», называются **линейными алгоритмами**.

Графическое представление алгоритмической конструкции «следование» приведено на рисунке:

На PascalABC.NET

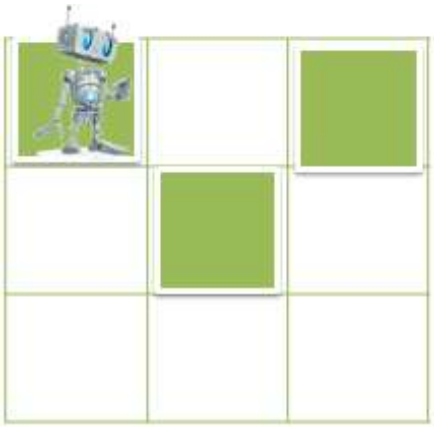
```
begin // гипотенуза
  var a := 3;
  var b := 4;
  var c := sqrt(a ** 2 + b ** 2);
  writeln(c);
end.
```

Линейный алгоритм приготовления отвара шиповника.

Обрати внимание, что многие из предписаний этого алгоритма могут потребовать детализации — представления в виде некоторой совокупности более мелких предписаний.

Пример:

*У исполнителя Робот есть четыре команды перемещения (вверх, вниз, влево и вправо), при выполнении каждой из них Робот перемещается на одну клетку в соответствующем направлении. По команде **закрасить** Робот закрашивает клетку, в которой он находится. Запишем линейный алгоритм, исполняя который Робот нарисует на клетчатом поле следующий узор и вернётся в исходное положение:*



```
алг узор
нач
  закрасить
  вправо
  вправо
  закрасить
  вниз
  влево
  закрасить
  вверх
  влево
кон
```

Пример:

Дан фрагмент линейного алгоритма:

```
x:=2
y:=x·x
y:=y·y
x:=y·x
s:=x+y
```

Шаг алгоритма	Переменные		
	x	y	s
1	2	-	-
2	-	4	-
3	-	16	-
4	32	-	-
5	-	-	48

С помощью операции **div** вычисляется целое частное, с помощью операции **mod** — остаток.

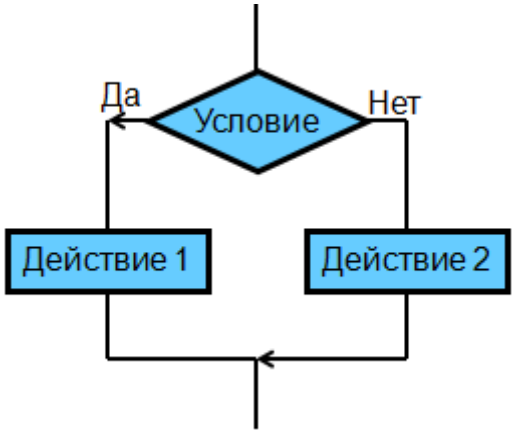
7 div 3 = 2	7 mod 3 = 1
8 div 3 = 2	8 mod 3 = 2
10 div 3 = 3	10 mod 3 = 1
13 div 4 = 3	13 mod 4 = 1

Ветвление

Ветвление — алгоритмическая конструкция, в которой, в зависимости от результата проверки условия («да» или «нет»), предусмотрен выбор одной из двух последовательностей действий (ветвей). *Иначе, условие – это утверждение, высказывание – логическая величина. Может принимать значение* истина (true) *или* ложь (false).

Алгоритмы, в основе которых лежит структура «ветвления», называют **разветвляющимися**. Блок-схемы ветвления представлены на рисунках.

Полная форма ветвления



На алгоритмическом языке команда ветвления записывается так:

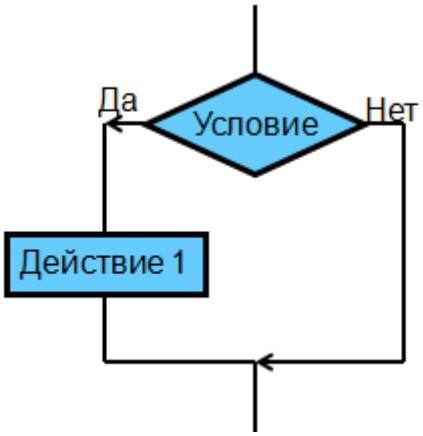
если <условие>
  то <действие 1>
  иначе <действие 2>
все

На PascalABC.NET

```
begin // находим модуль числа
  var x := -2018;
  if x < 0 then
    y := -x
  else
    y := x;
end.
```

алг правописание частиц НЕ, НИ
нач
  если частица под ударением
    то писать НЕ
    иначе писать НИ
  все
кон

Неполная форма ветвления



На алгоритмическом языке команда ветвления записывается так:

если <условие>
  то <действие 1>
все

На PascalABC.NET

```
begin // находим модуль числа
  var x := -2018;
  var y := x;
  if x < 0 then
    y := -x;
end.
```

алг сборы на прогулку
нач
  если на улице дождь
    то взять зонтик
  все
кон

Для записи условий, в зависимости от результатов проверки которых выбирается та или иная последовательность действий, используются операции сравнения:

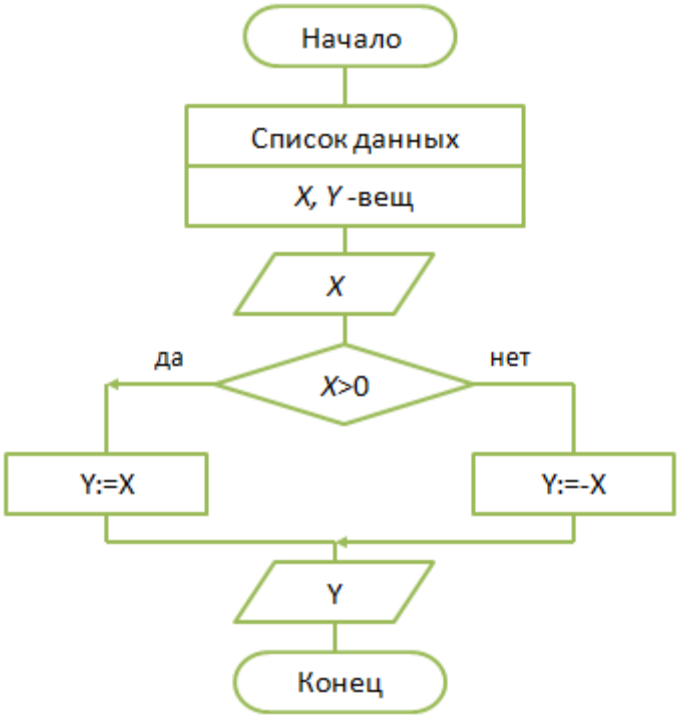
```
A<B     – A меньше B
A<=B    – A меньше или равно B
A=B     – A равно B
A>B     – A больше B
A>=B    – A больше или равно B
A<>B    – A не равно B
```

Здесь буквы A и B можно заменять на любые переменные, числа и арифметические выражения. Приведённые операции сравнения допускаются и для символьных переменных и текста.

Ветвление (задачи)

Пример:

Алгоритм вычисления функции $y(x)=|x|$ *для произвольного числа* x.



Обрати внимание на второй блок этой блок-схемы. В нём представлены имена и типы величин (данных), обрабатываемых в алгоритме.

Условия, состоящие из одной операции сравнения, называются простыми.

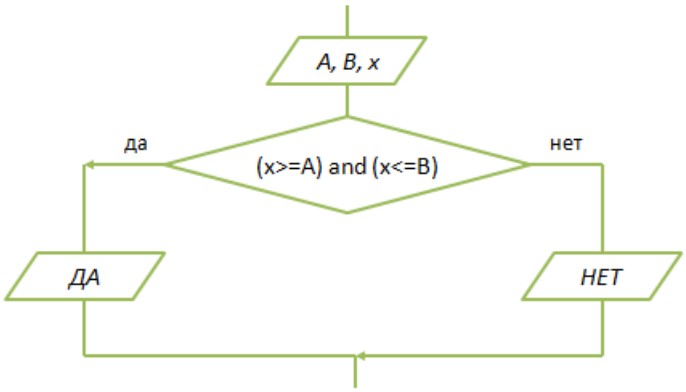
В качестве условий при организации ветвлений можно использовать и **составные условия**.

Обрати внимание!

Составные условия получаются из простых с помощью логических связок **and (и), or (или), not (не)**: and означает одновременное выполнение всех условий, or — выполнение хотя бы одного условия, а not означает отрицание условия, записанного за словом not.

Пример:

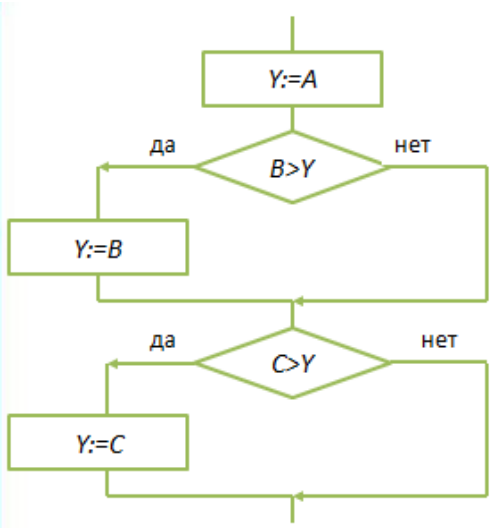
Алгоритм определения принадлежности точки x *отрезку* [a;b]. *Если точка* x *принадлежит данному отрезку, то выводится ответ ДА, в противном случае — НЕТ.*



Существует достаточно много ситуаций, в которых приходится выбирать не из двух, а из трёх и более вариантов. Есть разные способы построения соответствующих алгоритмов. Один из них составить комбинацию из нескольких ветвлений.

Пример:

Алгоритм, в котором переменной Y *присваивается значение большей из трёх величин* A, B *и* C.



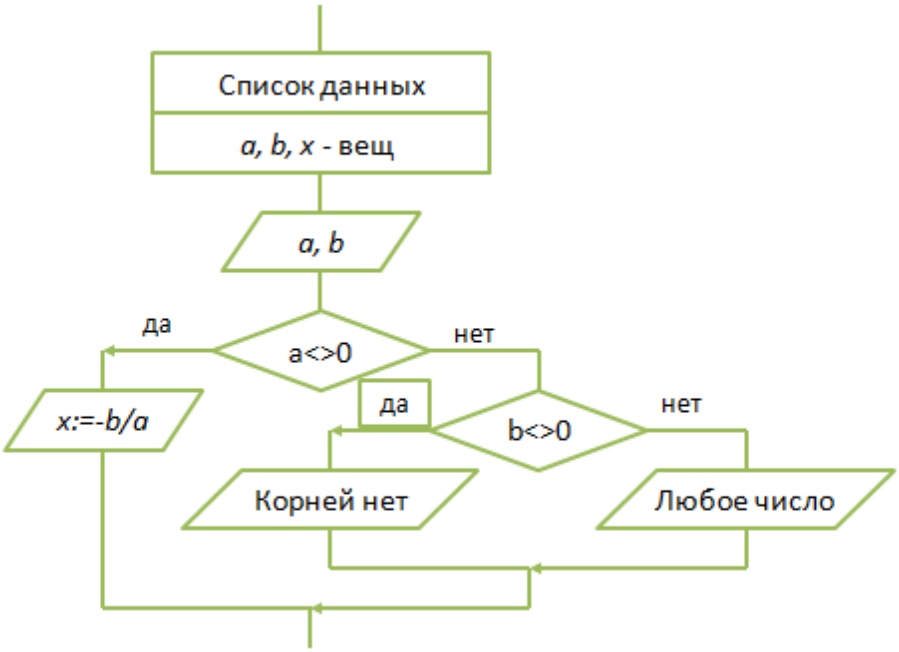
Пусть A=10, B=30 *и* C=20.
Тогда процесс выполнения алгоритма можно представить в следующей таблице:

Шаг	Константы			Переменная	Условие
	A	B	C	Y	
	10	30	20		
1				10	
2					30 > 10 (Да)
3				30	
4					20 > 30 (Нет)

Ответ: Y=30.

Пример:

Алгоритм решения линейного уравнения $ax+b=0$.



На PascalABC.NET

```
begin // линейное уравнение
  var a: real;
  var b: real;
  var x: real;
  a:= 2.5;
  b:= 10;
  if a <> 0 then
  begin
    x := b / a;
    writeln(x);
  end
  else
   if b <> 0 then
     writeln('Корней нет')
   else
     writeln('Любое число');
end.
```

Циклы

Повторение — алгоритмическая конструкция, представляющая собой последовательность действий, выполняемых многократно.

Алгоритмы, содержащие конструкцию повторения, называют **циклическими или циклами.**

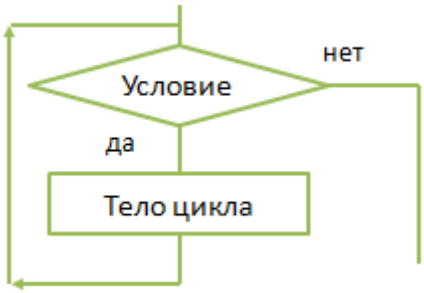
Последовательность действий, многократно повторяющаяся в процессе выполнения цикла, называется телом цикла.

В зависимости от способа организации повторений различают три типа циклов:

1. цикл с заданным условием продолжения работы;
2. цикл с заданным условием окончания работы;
3. цикл с заданным числом повторений.

Цикл с заданным условием продолжения работы

Логика работы этой конструкции описывается схемой, показанной на рисунке.



На алгоритмическом языке эта конструкция записывается так:

```
нц пока <условие>
<тело цикла (последовательность действий)>
кц
```

На PascalABC.NET

```
begin
  var i := 1;
  var s := 0;
  while i <= 5 do
  begin
    s += i; //увеличиваем значение на i
    i += 1; //увеличиваем значение на 1
  end;
  writeln(s);
end.
```

Выполняется **цикл-ПОКА** следующим образом:

- проверяется условие (вычисляется значение логического выражения);
- если **условие удовлетворяется (Да)**, то выполняется тело цикла и снова осуществляется переход к проверке условия;
- если же **условие не удовлетворяется**, то выполнение цикла заканчивается.

Возможны случаи, когда тело цикла не будет выполнено ни разу.

Пример:

Алгоритм, по которому из всех имеющихся кирпичей отбираются целые кирпичи и складываются в машину.

```
алг погрузка
 нач
  нц пока есть кирпичи
   взять один кирпич
   если кирпич целый
     то положить кирпич в машину
     иначе отложить кирпич в  сторону
   все
  кц
 кон
```

Пример:

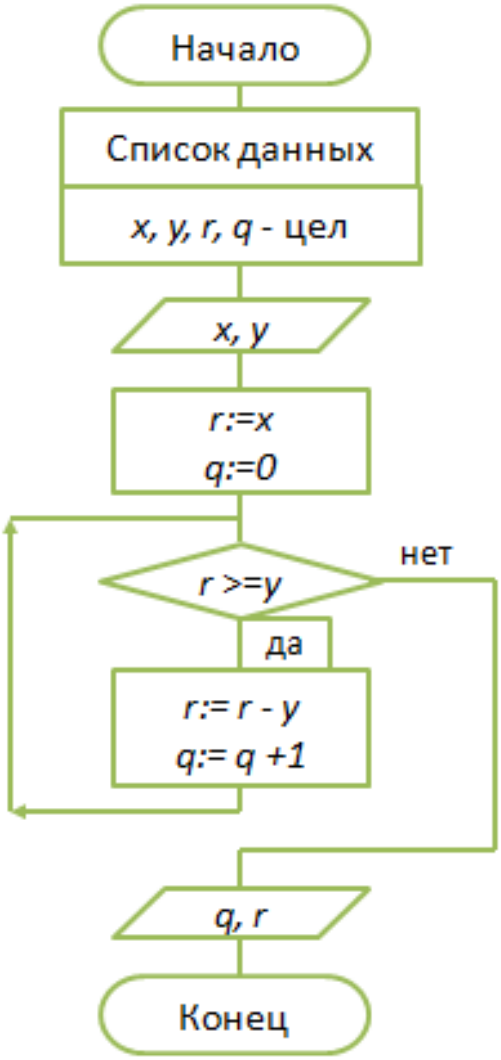
Правее Робота расположен коридор неизвестной длины. Необходимо, чтобы Робот закрасил все клетки этого коридора.

```
нц пока справа свободно
    вправо
    закрась
кц
```

Пример:

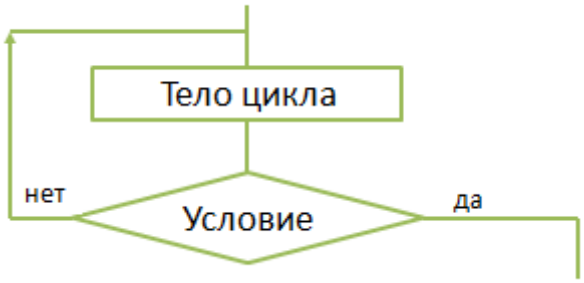
Требуется, не пользуясь операцией деления, получить частное q и остаток r от деления натурального числа x на натуральное число y.

Представим операцию деления как последовательные вычитания делителя из делимого. Причём вычитать будем до тех пор, пока результат вычитания не станет меньше вычитаемого (делителя). В этом случае количество вычитаний будет равно частному от деления q, а последняя разность — остатку от деления r.



Цикл с заданным условием окончания работы

Логика работы этой конструкции описывается схемой, показанной на рисунке.



На алгоритмическом языке эта конструкция записывается так:

нц
<тело_цикла (последовательность действий)>
кц при <условие>

На PascalABC.NET

```
begin
  var i := 1;
  var s := 0;
  repeat
    s += i;
    i += 1; //увеличиваем значение на 1
  until i > 5;
  writeln(s);
end.
```

Выполняется **цикл-ДО** следующим образом:

- выполняется тело цикла;
- проверяется условие (вычисляется значение логического выражения); если условие не удовлетворяется («Нет»), то снова выполняется тело цикла и осуществляется переход к проверке условия;
- если же условие удовлетворяется, то выполнение цикла заканчивается.

В любом случае тело цикла будет выполнено хотя бы **один раз.**

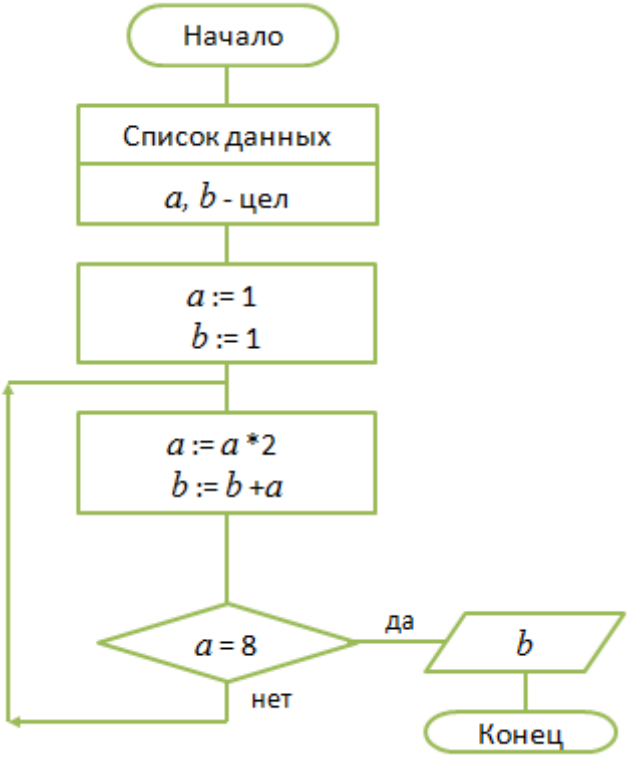
Пример:

Алгоритм по выучиванию наизусть четверостишия.

алг четверостишие
нач
нц
прочитать четверостишие по книге 1 раз
прочитать четверостишие наизусть
кц при не сделал ошибку
кон

Пример:

Вычислим значение переменной b *согласно следующему алгоритму:*

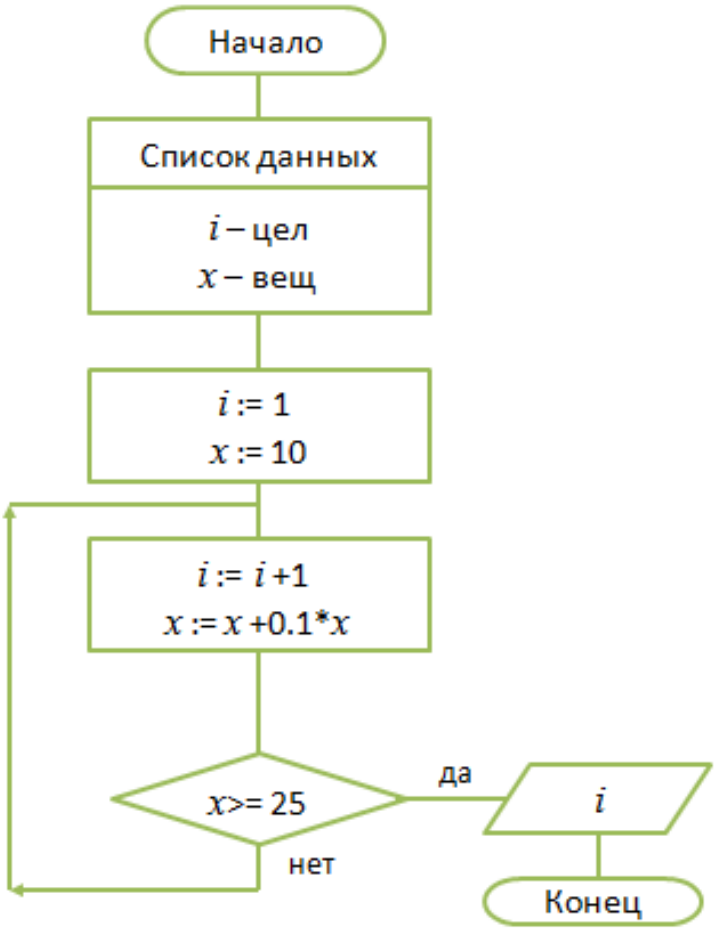


Шаг алгоритма	Операция	Переменные		Условие
		a	b	a = 8
1	a := 1	1		
2	b := 1	1	1	
3	a := a * 2	2	1	
4	b := b+a	2	3	
5	a = 8			2 = 8 (Нет)
6	a := a * 2	4	3	
7	b := b+a	4	7	
8	a = 8			4 = 8 (Нет)
9	a := a * 2	8	7	
10	b := b+a	8	15	
11	a = 8			8 = 8 (Да)

Пример:

Спортсмен приступает к тренировкам по следующему графику: в первый день он должен пробежать 10 км; каждый следующий день следует увеличивать дистанцию на 10% от нормы предыдущего дня. Как только дневная норма достигнет или превысит 25 км, необходимо прекратить её увеличение и далее пробегать ежедневно ровно 25 км. Начиная с какого дня спортсмен будет пробегать 25 км?

Пусть x — количество километров, которое спортсмен пробежит в некоторый i-й день. Тогда в следующий (i+1)-й день он пробежит x+0,1x километров (0,1x — это 10% от x).



На PascalABC.NET

```
begin // спортсмен
  var i := 1;
  var x := 10.0;
  repeat
    x += 0.1 * x;
    i += 1;
  until x >= 25;
  writeln(i);
end.
```

Цикл с заданным числом повторений

Логика работы этой конструкции описывается схемой, показанной на рисунке.

На алгоритмическом языке эта конструкция записывается так:

нц для i **от** i1 **до** i2 **шаг** R
<тело_цикла (последовательность действий)>
кц

На PascalABC.NET

```
begin
  var s:=0;
  for var i:=1 to 5 do
  begin
    s +=i;
  end;
  writeln(s);
end.
```

Обрати внимание!

В **цикле-ДЛЯ** всегда есть параметр цикла — величина целого типа, изменяющаяся в ходе выполнения цикла от своего начального значения i1 до конечного значения i2 с шагом R.

Выполняется **цикл-ДЛЯ** следующим образом:

- параметру цикла присваивается начальное значение;
- параметр цикла сравнивается с конечным значением; если параметр цикла не превышает конечное значение, то выполняется тело цикла, увеличивается значение параметра цикла на шаг и снова осуществляется проверка параметра цикла; если же параметр цикла превышает конечное значение, то выполнение цикла заканчивается.

Если величина шага в цикле с параметром равна единице, то шаг не указывают. Мы ограничимся рассмотрением именно таких циклов. В отличие от двух предыдущих конструкций (цикл-ПОКА, цикл-ДО) цикл-ДЛЯ имеет строго фиксированное число повторений, что позволяет избежать зацикливания, т.е. ситуации, когда тело цикла выполняется бесконечно.

Пример:

Алгоритм переправы через реку воинского отряда из пяти человек. Солдаты могут воспользоваться помощью двух мальчиков — хозяев небольшой лодки, в которой может переправиться или один солдат, или два мальчика.

алг переправа
нач
нц для i **от** 1 **до** 5
два мальчика переправляются на противоположный берег
один мальчик высаживается на берег
другой мальчик плывёт обратно
солдат переправляется через реку
мальчик возвращается на исходную позицию
кц
кон

Для исполнителя Робот цикл с известным числом повторений реализуется с помощью следующей конструкции:

нц <число повторений> **раз**
<тело цикла>
кц

Пример:

Так, если правее Робота не встретится препятствий, то, выполнив приведённый ниже алгоритм, он переместится на пять клеток вправо и закрасит эти клетки:

алг
нач
нц 5 раз
вправо; закрасить
кц
кон