Цикл «пока»

Решим задачу.

З а д а ч а. На поле где-то ниже *Робота* на неизвестном расстоянии есть стена. Нужно, чтобы *Робот* подошёл вплотную к этой стене.

Если мы командуем *Роботом* вручную, без компьютера, то задачу можно решить так: дадим *Роботу* запрос `снизу свободно`. Если *Робот* ответит «нет», значит, он уже у стены и задача решена. Если же *Робот* ответит «да», то можно скомандовать `вниз` и опять дать запрос `снизу свободно`. Если *Робот* ответит «да» — опять скомандовать `вниз`, дать запрос `снизу свободно`, и так поступать дальше, пока *Робот* не ответит «нет». Заранее неизвестно, сколько раз мы дадим команду `вниз` — 0, 3, 8 или 2014 раз, это зависит от начального расположения *Робота* относительно стены. Например, если *Робот* уже стоял рядом со стеной, то команда `вниз` не подавалась бы ни разу. Если *Робот* стоял в 101-й клетке от стены, то команда `вниз` подавалась бы 100 раз.

Наши действия при решении этой задачи можно описать так: пока на запрос `снизу свободно` *Робот* отвечает «да», надо командовать `вниз` и повторять запрос. Такое описание пригодно при ***любом*** расстоянии от *Робота* до стены.

Наша цель, однако, не ручное управление *Роботом,* а составление программы (алгоритма) для компьютера. Поэтому приведённую выше последовательность действий надо описать на алгоритмическом языке. Алгоритм должен быть универсальным, т. е. не должен зависеть от расстояния между *Роботом* и стеной. Для этого в алгоритмическом языке есть специальная *составная команда* — **цикл «пока»**. В общем виде **цикл «пока»** записывается так:

```
нц пока <условие>
  <последовательность команд цикла>
кц
```

Слова **нц** и **кц** имеют тот же смысл, что и в **цикле «N раз»**, — они отмечают начало и конец цикла. При выполнении **цикла «пока»** компьютер повторяет следующие действия:

а) проверяет *условие,* записанное после служебного слова **пока**;

б) если условие не соблюдается (*Робот* ответил «нет»), то выполнение цикла завершается и компьютер начинает выполнять команды, записанные после **кц**. Если же условие соблюдается (*Робот* ответил «да»), то компьютер выполняет последовательность команд цикла, потом снова проверяет условие и т. д.

Вот алгоритм вниз до стены для решения задачи, в нём использована составная команда **цикл «пока»:**

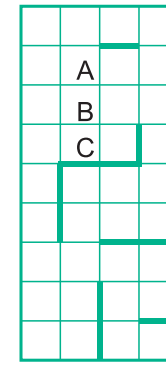
```
алг вниз до стены
  дано |
  надо | Робот идёт вниз,
       | пока не дойдёт
       | до стены
нач
  нц пока снизу свободно
    вниз
  кц
кон
```

Аналогично можно составить алгоритмы влево до стены, вправо до стены, вверх до стены, которыми мы далее часто будем пользоваться.

Как *Робот* выполняет такой алгоритм?

Например, пусть *Робот* стоит в клетке A. Тогда при выполнении алгоритма вниз до стены диалог между *Роботом* и компьютером будет таким:

```
компьютер: снизу свободно?
Робот: да
компьютер: вниз
Робот: <смещается вниз в клетку B>
компьютер: снизу свободно?
Робот: да
компьютер: вниз
Робот: <смещается вниз в клетку C>
компьютер: снизу свободно?
Робот: нет
```



Поскольку *Робот* ответил «нет», т. е. записанное после **пока** условие не соблюдается, то на этом выполнение **цикла «пока»** и алгоритма вниз до стены заканчивается.

Свойства цикла «пока»

Свойство 1. *Последовательность команд цикла может не выполниться ни разу.*

Если условие в **цикле «пока»** не соблюдается с самого начала, то последовательность команд цикла не выполняется ни разу. Например, если в алгоритме вниз до стены *Робот* на первый же запрос снизу свободно ответит «нет» (т. е. если по нижней границе клетки, в которой находится *Робот*, с самого начала уже стоит стена), то компьютер не вызовет команду вниз ни разу. Важно понимать, что ситуация, когда цикл ни разу не выполняется, не является *отказом*. Это нормальный вариант выполнения алгоритма.

Свойство 2. *Выполнение цикла может не завершиться.*

Выполнение **цикла «пока»** может не завершиться, если условие всё время будет соблюдаться. Такая ситуация обычно возникает из-за ошибок в составлении алгоритма. Она называется *зацикливанием*.

Например, рассмотрим такой фрагмент алгоритма:

```
нц пока снизу свободно
  вниз
  вверх
кц
```

Если снизу от *Робота* нет стены, он будет *бесконечно* топтаться на месте, совершая шаги вверх и вниз. Исполнение алгоритма никогда не закончится.

Свойство 3. *Условие цикла не проверяется в процессе выполнения последовательности команд цикла.*

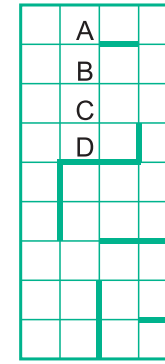
Условие в **цикле «пока»** проверяется только перед выполнением последовательности команд цикла, но не проверяется в процессе выполнения этих команд.

Пример. Пусть *Робот* находится в клетке A (см. рисунок) и компьютер выполняет такой цикл:

```
нц пока снизу свободно
  вниз
  вниз
кц
```

Тогда диалог между компьютером и *Роботом* будет таким:

```
компьютер: снизу свободно?
Робот: да
компьютер: вниз
Робот: <смещается вниз в клетку B>
компьютер: вниз
Робот: <смещается вниз в клетку C>
компьютер: снизу свободно?
Робот: да
компьютер: вниз
Робот: <смещается вниз в клетку D>
компьютер: вниз
Робот: ОТКАЗ
```



В процессе выполнения последовательности команд цикла компьютер не проверяет, истинно ли условие выполнения цикла. Если после выполнения какой-то команды внутри **цикла «пока»** условие перестанет соблюдаться, то компьютер всё равно будет выполнять последовательность команд цикла до конца.

Свойство 4. *Перед каждым выполнением последовательности команд цикла условие обязательно выполняется.*

Это очевидное свойство: если компьютер приступает к выполнению последовательности команд цикла, значит, условие выполнено.

Свойство 5. *Сразу после окончания цикла условие не выполняется.*

Это свойство тоже очевидно: цикл заканчивается в тот момент, когда при очередной проверке условие оказалось невыполненным.

Составление алгоритма с циклом «пока»

Цикл «пока» используется всякий раз, когда число повторений каких-то действий заранее неизвестно. Составление таких циклов — трудная задача.

Чтобы не запутаться и что-нибудь не забыть, лучше всего строить **цикл «пока»** по частям:

1. Понять, когда цикл должен закончиться, т. е. сформулировать условие выполнения цикла. Иногда это удобно делать от противного — записать противоположное условие.
2. Описать, что происходит при однократном выполнении цикла (принято говорить «за один шаг цикла»), т. е. *выполнить один раз последовательность команд цикла.*
3. Проверить, что цикл рано или поздно закончится, а не будет повторяться вечно.

Циклы **«N раз»** и **«пока»** оформляются в алгоритмическом языке почти одинаково — обе эти команды задают повторяющуюся последовательность команд. Однако у этих двух циклов есть одно существенное различие. Начиная выполнять **цикл «N раз»**, компьютер *знает*, сколько раз придётся повторить последовательность команд цикла. При исполнении **цикла «пока»** это не так: компьютер каждый раз проверяет условие цикла и *не может заранее определить*, через сколько повторений выполнение закончится. Определить количество повторений **цикла «пока»** можно только после того, как цикл завершён.

Отсюда ясно, в каких случаях какой цикл следует использовать. Если к моменту начала цикла количество повторений известно, можно воспользоваться **циклом «N раз»**. Если же количество повторений заранее определить нельзя, необходим **цикл «пока»**.

141 Составь диалог компьютера и *Робота* при выполнении каждого цикла в ситуации, когда *Робот* стоит: а) в закрашенной клетке; б) в незакрашенной клетке. Если выполнение алгоритма приведёт к зацикливанию — прерви описание диалога и напиши о зацикливании.

```
нц пока клетка чистая
  закрасить
кц
```

```
нц пока клетка закрашена
  закрасить
кц
```

142 Составь алгоритмы влево до стены, вправо до стены, вверх до стены, о которых идёт речь на с. 88. Запиши условия **дано** и **надо.** Проверь, что алгоритмы составлены правильно — что не возникнет отказа и что цикл не будет повторяться вечно.

143 *Робот* находится у левой стены прямоугольного поля, огороженного со всех сторон стенами. Внутри поля стен нет, размеры поля неизвестны. Надо составить такой алгоритм, при выполнении которого *Робот* закрасит горизонтальный ряд клеток от исходного положения до правой стены и вернётся в исходное положение: сначала *Робота* надо довести до правой стены, по дороге закрашивая клетки, а затем вернуть в исходное положение и закрасить эту исходную клетку. Вот этот алгоритм:

```
алг закрасить ряд вправо и вернуться
  дано | Робот стоит у левой стены поля
  надо | закрашен весь горизонтальный ряд,
       | в котором стоит Робот,
       | Робот в исходном положении
нач
  вправо до стены с закрашиванием
  влево до стены
  закрасить
кон
```

Вспомогательный алгоритм влево до стены ты построил в ходе решения задачи 142. Построй вспомогательный алгоритм вправо до стены с закрашиванием. Запиши условия **дано** и **надо.** Проверь, что алгоритм составлен правильно — что не возникнет отказа и что цикл не будет повторяться вечно.

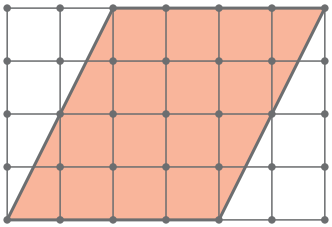
144 Составь алгоритм с таким заголовком (см. ниже).

```
алг закрасить прямоугольник
  дано | Робот стоит в левом нижнем углу
       | поля без внутренних стен
  надо | весь прямоугольник закрашен,Робот
       | в исходном положении
```

Используй в качестве вспомогательного алгоритм закрасить ряд вправо и вернуться, составленный при решении задачи 143.

145 Реши задачу 144, считая, что о начальном положении *Робота* на поле ничего не известно.

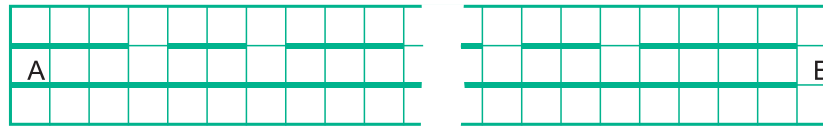
146 Нарисуй такой же четырёхугольник по клеткам в тетради. Потом нарисуй, как разрезать его на части, чтобы из них можно было собрать квадрат на сетке. Нарисуй такой квадрат и покажи штриховыми линиями, из каких частей он составлен.



147 На поле *Робота* ровно одна стена. Это горизонтальная стена, которая не примыкает к границам поля. *Робот* находится в одной из клеток, прилегающих к стене сверху. Точные размеры поля и стены и точное расположение *Робота* неизвестны. Составь алгоритм, при выполнении которого *Робот:*

а) окажется в одной из клеток, прилегающих к стене снизу;
б) закрасит все клетки, прилегающие к стене снизу;
в) закрасит все прилегающие к стене клетки.

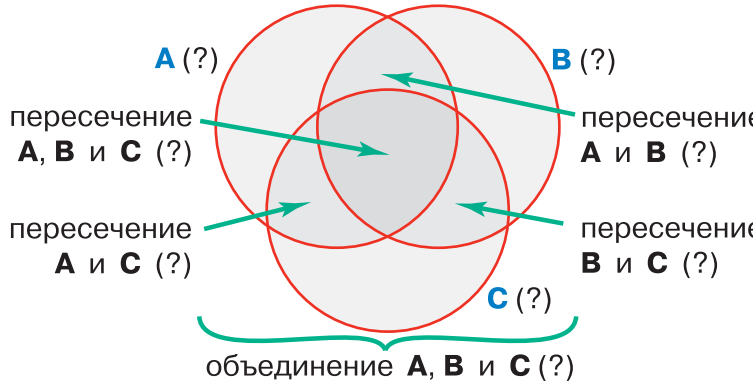
148 *Робот* стоит в левом конце горизонтального коридора, нижняя стена которого сплошная, а в верхней имеется несколько выходов. Составь алгоритм, который заставляет *Робота* переместиться из клетки A в клетку B (см. рисунок, сколько клеток между клетками A и B, неизвестно) и закрасить все те клетки коридора, которые расположены напротив боковых выходов.



149 Реши задачу.

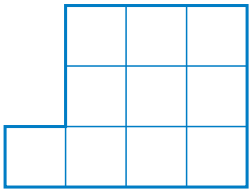
В нашем классе из 31 учащегося по итогам первой четверти неуспевающих нет (ни у кого нет ни одной четвертной двойки). При этом у нас имеется 3 круглых отличника и 3 круглых троечника (тех, у кого тройки в четверти по всем предметам). Пятеро учеников закончили четверть только на четвёрки и пятёрки. Известно также, что пятёрки имеют среди четвертных оценок 18 учеников и 24 ученика имеют среди четвертных оценок четвёрки. Сколько учеников нашего класса имеют в четверти и тройки, и четвёрки, и пятёрки? Сколько учеников нашего класса не получили в четверти ни одной четвёрки?

Для решения задачи нарисуй схему с множествами:



150 Нарисуй в тетради по клеткам:
а) прямоугольник, не являющийся квадратом, площадь которого равна 64 ед. кв.;
б) прямоугольный треугольник, площадь которого равна 17 ед. кв.;
в) четырёхугольник, площадь которого равна 13 ед. кв.;
г) два разных прямоугольных треугольника, площадь каждого из которых равна $10\frac{1}{2}$ ед. кв.;
д) два разных четырёхугольника, не являющихся прямоугольниками, площадь каждого из которых равна 21 ед. кв.

151 К полю для игры *Крестики-нолики* добавлена одна клетка. Правила игры при этом остались прежними: Первый ставит крестики, Второй — нолики. Выигрывает тот, кто первым построит ряд из трёх своих значков по горизонтали, вертикали или диагонали. В отличие от игры на обычном поле, в игре *Крестики-нолики* на таком поле Первый имеет выигрышную стратегию. Куда должен поставить крестик Первый на первом ходу, чтобы наверняка выиграть?



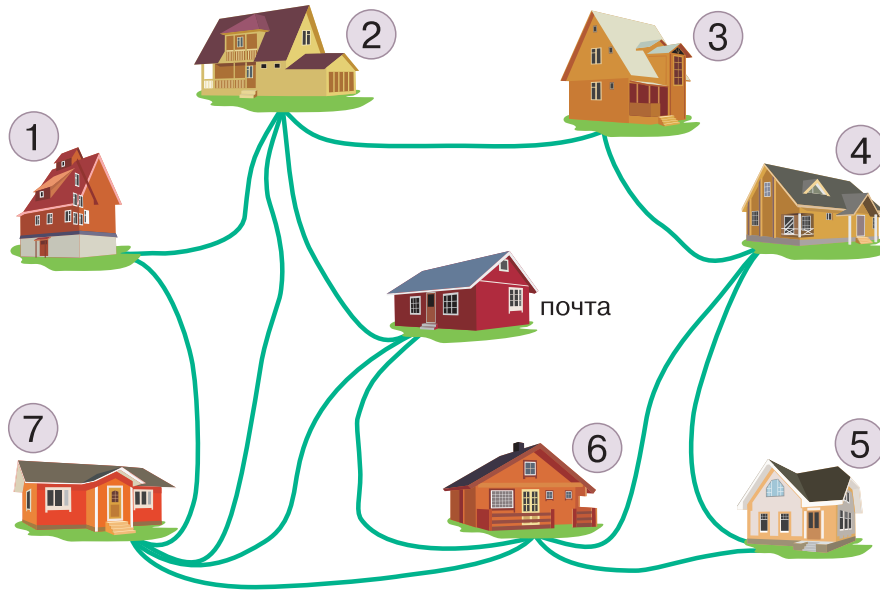
152 Вот правила игры *7 камней*:

> **Правила игры *7 камней***
>
> Начальная позиция. Куча из 7 камешков.
> Возможные ходы. На каждом ходу игрок разделяет любую из имеющихся куч на две (необязательно пополам).
> Как определить победителя. Игра заканчивается, если очередной ход сделать невозможно. Выигрывает тот, кто сделал последний ход.

Исследуй игру *7 камней.* Попытайся объяснить, почему в этой игре выигрышная стратегия не нужна, — победа не зависит от того, насколько умело играют игроки. Кто выигрывает в игре *7 камней*?

153 Реши задачу.

Почтальон Печкин, выйдя из почтового отделения, разнёс почту в каждый дом деревни, после чего зашёл с посылкой к дяде Фёдору (зайдя сначала за ней на почту), а потом вернулся домой. На рисунке показаны все тропинки, по которым проходил Печкин, причём, как оказалось, ни по одной из них он не проходил дважды. Каков мог быть маршрут почтальона Печкина? В каком доме живёт дядя Фёдор?



Равновесные выигрышные стратегии

До сих пор, для того чтобы построить выигрышную стратегию, мы исследовали все позиции игры и выделяли выигрышные и проигрышные позиции. Но для многих игр существуют стратегии, построение которых не требует исследования всех позиций. Таковы, например, *равновесные* стратегии, которыми мы сейчас займёмся.