Циклические алгоритмы

Как организовать цикл?

Цикл – это многократное выполнение одинаковых действий. Доказано, что любой алгоритм может быть записан с помощью трёх алгоритмических конструкций: циклов, условных операторов и последовательного выполнения команд (линейных алгоритмов).

Подумаем, как можно организовать цикл, который 10 раз выводит на экран слово «привет». Вы знаете, что программа после запуска выполняется процессором автоматически. И при этом на каждом шаге нужно знать, сколько раз уже выполнен цикл и сколько ещё осталось выполнить. Для этого необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла (счётчик шагов). Сначала можно записать в неё ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение ячейки на единицу. На псевдокоде алгоритм можно записать так (здесь и далее операции, входящие в тело цикла, выделяются отступами):

```
счётчик = 0
пока счётчик != 10
  print ( "Привет!" )
  увеличить счётчик на 1
```

Возможен и другой вариант: сразу записать в счётчик нужное количество шагов, и после каждого шага цикла *уменьшать* счётчик на 1. Тогда цикл должен закончиться при нулевом значении счётчика:

```
счётчик = 10
пока счётчик > 0
  print ( "Привет!" )
  уменьшить счётчик на 1
```

Этот вариант несколько лучше, чем предыдущий, поскольку счётчик сравнивается с нулём, а такое сравнение выполняется в процессоре автоматически (см. главу 4).

В этих примерах мы использовали *цикл с условием*, который выполняется до тех пор, пока некоторое условие не становится ложно.

Циклы с условием

Рассмотрим следующую задачу: определить количество цифр в десятичной записи целого положительного числа. Будем предполагать, что исходное число записано в переменную **n** целого типа.

Сначала нужно разработать алгоритм решения задачи. Чтобы подсчитывать что-то в программе, нужно использовать переменную, которую называют *счётчиком*. Для подсчёта количества цифр нужно как-то отсекать эти цифры по одной, с начала или с конца, каждый раз увеличивая счётчик. Начальное значение счётчика должно быть равно нулю, так как до выполнения алгоритма ещё не найдено ни одно цифры.

Для отсечения первой цифры необходимо заранее знать, сколько цифр в десятичной записи числа, то есть нужно заранее решить ту задачу, которую мы решаем. Следовательно, этот метод не подходит.

Отсечь последнюю цифру проще – достаточно разделить число нацело на 10 (поскольку речь идет о десятичной системе). Операции отсечения и увеличения счётчика нужно выполнять столько раз, сколько цифр в числе. Как же «поймать» момент, когда цифры кончатся? Несложно понять, что в этом случае результат очередного деления на 10 будет равен нулю, это и говорит о том, что отброшена последняя оставшаяся цифра. Изменение переменной **n** и счётчика для начального значения 1234 можно записать в виде таблицы, показанной справа. Псевдокод выглядит так:

n	счётчик
1234	0
123	1
12	2
1	3
0	4

```
счётчик = 0
пока n > 0
  отсечь последнюю цифру n
  увеличить счётчик на 1
```

Программа на Python выглядит так:

```
count = 0
while n > 0:
  n = n // 10
  count += 1
```

Слово **while** переводится как «пока», то есть, цикл выполняется пока **n > 0**. Переменная-счётчик имеет имя **count**.

Обратите внимание, что проверка условия выполняется в начале очередного шага цикла. Такой цикл называется *циклом с предусловием* (то есть с предварительной проверкой условия) или циклом «пока». Если в начальный момент значение переменной **n** будет нулевое или отрицательное, цикл не выполнится ни одного раза.

В данном случае количество шагов цикла «пока» неизвестно, оно равно количеству цифр введенного числа, то есть зависит от исходных данных. Кроме того, этот же цикл может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k = 0
while k < 10:
  print ( "привет" )
  k += 1
```

Если условие в заголовке цикла никогда не нарушится, цикл будет работать бесконечно долго. В этом случае говорят, что «программа зациклилась». Например, если забыть увеличить переменную **k** в предыдущем цикле, программа зациклится:

```
k = 0
while k < 10:
  print ( "привет" )
```

Во многих языках программирования существует *цикл с постусловием*, в котором условие проверяется после завершения очередного шага цикла. Это полезно в том случае, когда нужно обязательно выполнить цикл хотя бы один раз. Например, пользователь должен ввести с клавиатуры положительное число и защитить программу от неверных входных данных.

В языке Python нет цикла с постусловием, но его можно организовать с помощью цикла **while**:

```
print ( "Введите положительное число:" )
n = int ( input() )
while n <= 0:
  print ( "Введите положительное число:" )
  n = int ( input() )
```

Однако такой вариант не очень хорош, потому что нам пришлось написать два раза пару операторов. Но можно поступить иначе:

```
while True:
  print ( "Введите положительное число:" )
  n = int ( input() )
  if n > 0: break
```

Цикл, который начинается с заголовка **while True** будет выполняться бесконечно, потому что условие **True** всегда истинно. Выйти из такого цикла можно только с помощью специального оператора **break** (в переводе с англ. – «прервать», досрочный выход из цикла). В данном случае он сработает тогда, когда станет истинным условие **n > 0**, то есть тогда, когда пользователь введет допустимое значение.

Цикл с переменной

Вернёмся снова к задаче, которую мы обсуждали в начале параграфа – вывести на экран 10 раз слово «привет». Фактически нам нужно организовать цикл, в котором блок операторов выполнится заданное число раз (в некоторых языках такой цикл есть, например, в школьном алгоритмическом языке он называется «цикл N раз»). На языке Python подобный цикл записывается так:

```
for i in range(10):
  print ( "Привет!" )
```

Здесь слово **for** означает «для», переменная **i** (её называют переменной цикла) изменяется в диапазоне (**in range**) от 0 до 10, *не включая* 10 (то есть от 0 до 9 включительно). Таким образом, цикл выполняется ровно 10 раз.

В информатике важную роль играют степени числа 2 (2, 4, 8, 16 и т.д.) Чтобы вывести все степени двойки от 2^1 до 2^{10} мы уже можем написать такую программу с циклом «пока»:

```
k = 1
while  k <= 10 :
  print ( 2**k )
  k += 1
```

Вы наверняка заметили, что переменная **k** используется трижды (см. выделенные блоки): в операторе присваивания начального значения, в условии цикла и в теле цикла (увеличение на 1). Цикл с переменной «собирает» все действия с ней в один оператор:

```
for k in range(1,11):
  print ( 2**k )
```

Здесь диапазон (**range**) задается двумя числами – начальным и конечным значением, причем указанное конечное значение *не входит* в диапазон. Такова особенность функции **range** в Python.

Шаг изменения переменной цикла по умолчанию равен 1. Если его нужно изменить, указывают третье (необязательное) число в скобках после слова **range** – нужный шаг. Например, такой цикл выведет только нечётные степени числа 2 (2^1, 2^3 и т.д.):

```
for k in range(1,11,2):
  print ( 2**k )
```

С каждым шагом цикла переменная цикла может не только увеличиваться, но и уменьшаться. Для этого начальное значение должно быть больше конечного, а шаг – отрицательный. Следующая программа печатает квадраты натуральных чисел от 10 до 1 в порядке убывания:

```
for k in range(10,0,-1):
  print ( k**2 )
```

Вложенные циклы

В более сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция «цикл в цикле» или «вложенный цикл».

Предположим, что нужно найти все простые числа в интервале от 2 до 1000. Простейший (но не самый быстрый) алгоритм решения такой задачи на псевдокоде выглядит так:

```
for n in range(2,1001):
  if число n простое:
    print ( n )
```

Как же определить, что число простое? Как известно, простое число делится только на 1 и само на себя. Если число **n** не имеет делителей в диапазоне от 2 до **n-1**, то оно простое, а если хотя бы один делитель в этом интервале найден, то составное.

Чтобы проверить делимость числа **n** на некоторое число **k**, нужно взять остаток от деления **n** на **k**. Если этот остаток равен нулю, то **n** делится на **k**. Таким образом, программу можно записать так (здесь **n**, **k** и **count** – целочисленные переменные, **count** обозначает счётчик делителей):

```
for n in range(2,1001):
  count = 0
  for k in range(2,n):
    if n%k == 0:
      count += 1
  if count == 0:
    print ( n )
```

Попробуем немного ускорить работу программы. Делители числа обязательно идут в парах, причём в любой паре меньший из делителей не превосходит $\sqrt{n}$ (иначе получается, что произведение двух делителей, каждый из которых больше $\sqrt{n}$, будет больше, чем **n**). Поэтому внутренний цикл можно выполнять только до значения $\sqrt{n}$ вместо **n-1**. Для того, чтобы работать только с целыми числами (и таким образом избежать вычислительных ошибок), лучше заменить условие $k \le \sqrt{n}$ на равносильное ему условие $k^2 \le n$. При этом потребуется перейти к внутреннему циклу с условием:

```
count = 0
k = 2
while k*k <= n:
  if n%k == 0:
    count += 1
  k += 1
```

Чтобы еще ускорить работу цикла, заметим, что когда найден хотя бы один делитель, число уже заведомо составное, и искать другие делители в данной задаче не требуется. Поэтому можно закончить цикл. Для этого при **n%k == 0** выполним досрочный выход из цикла с помощью оператора **break**, причём переменная **count** уже не нужна:

```
k = 2
while k*k <= n:
  if n%k == 0: break
  k += 1
if k*k > n:
  print ( n )
```

Если после завершения цикла **k*k>n** (нарушено условие в заголовке цикла), то число **n** простое.

В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
for i in range(1,5):
  for k in range(1,i+1):
    print ( i, k )
```

На первом шаге (при **i=1**) переменная **k** принимает единственное значение 1. Далее, при **i=2** переменная **k** принимает последовательно значения 1 и 2. На следующем шаге при **i=3** переменная **k** проходит значения 1, 2 и 3, и т.д.