# Циклы

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

16 декабря 2016 г.

Версия: 0.15

**Аннотация**

Как выполнить неоднократно в зависимости от некого условия отличные от основной программы действия.

Цель: Необходимость в обработки произвольного количества данных, а также вычисление некой последовательности по индукции, в частности, поиск некого значения с необходимой точностью.

Предварительная версия.

## 1 Циклы

Необходимость в многократном выполнении части команд, более того, часто при заранее неизвестном их количестве.

Для обработки заранее не обговоренного количества данных. В частности, вычисление суммы введенных чисел.

Многократное выполнение определенного действия, пока будет выполнено или не выполнено некое условие. В частности, поиск нужного члена последовательности по индукции.

В первом подразделе описывается применение циклов. В втором подразделе дается продвинутое описание логических выражений, которое позволяет лаконичнее записывать код программы. В третьем – рассмотрены разновидности частных конструкций циклов.

### 1.1 Цикл while (пока)

Цикла *пока* самый общий. Им можно задать любые другие виды циклов. Так, при написании программ можно им только и пользоваться.

**Языковая конструкция** `while` Цикл `while` – языковая конструкция языка Си, которая позволяет выполнять команду (см. 9) пока истинно соответствующее (см. 8) основное условие.

```
1  //while( логическое выражение )//проверяемое условие
2  //      команда;//выполняем, пока истинно выражение.
3  //
4  int s, a;
5  printf("enter a:");//приглашение на ввод
6  scanf("%d", &a);//считываем значение
7  s=i=0;//Присвоение s и i значение 0.
8  while( s < a )//Выполнять пока истинно данное условие.
9          s = s + 1;//Выполняемое действие.
10 printf("summa ~ %d\n", s);
```

Обращаю внимание на отсутствие символа конца команды – символа `;` в конце строчки с `while` (см. стр. 8). Данный символ, если и ставится, то только в конце целевой команды (см. стр. 9). Так, он в обще может отсутствовать.

Упражнение. Поиграйся с кодом. Замени число 1, на 2, на 3, на 0. В последнем случае пригодится комбинация Ctrl-C для принудительного завершения программы при работе в консоли. Пойми правильность результата работы, в том числе числового, программы.

Упражнение. Что будет, если убрать присвоение 0я переменным `s` и `i`? (см. стр. 7)

В данном фрагменте кода показаны главные составные части при написании цикла. К написанию цикла можно отнестись как к методу индукции, а именно – первый шаг (инициализация переменных) и шаг, обеспечивающий шаг индукции. При выполнении этих свойств цикл корректен. В дополнении к этому в программе есть проверка на необходимость продолжения индукции (*условие цикла*): иначе говоря задает момент завершения цикла.

На первом шаге производится инициализация переменных, а именно – задание части переменным первоначального (до момента входа в цикл) значения. В противном случае, нет гарантии, что "индукция"будет успешно выполнена. В частности, если в данном фрагменте убрать (см. стр. 7) присвоение 0 переменной s, то как понимать строчку 9 при первом выполнении целевого действия цикла? Как к новому значению прибавить предыдущие, если оно ещё не было задано? Так вот, по этой причине перед началом цикла и необходимо задавать значения всем актуальным переменным (см. стр. 7). В данном случае, переменные нужно положить равными 0.

Шаг индукции, который соответствует сути цикла (см. стр. 9), пишется из предположения, что все актуальные переменные имеют корректное значение и

их можно использовать в арифметических и логических выражениях. В данном случае, если действительно перед началом цикла присвоить значение 0 переменной **s**, то все встает на свои места, так как при первом проходе цикла в стр. 9 старое значение переменной **s** будет равным 0.

Как и в условных конструкция, одиночную команду естественно можно заменить на фрагмент когда:

```
1  int s=0, i=0, a;//Можно инициализировать сразу.
2  printf("enter a:");//приглашение на ввод
3  scanf("%d", &a);//считываем значение
4  while( s < a )
5  {//Выполнять пока истинно данное условие.
6          i = i + 1;
7          s = s + i;//Выполняемое действие.
8  }
9  //Раз из цикла вышли, значит нарушено условие цикла.
10 //т.е. s>=a.
11 printf("summa ~ %d, iter ~ %d\n", s, i);
```

Цикл выполняется пока истинно основное условие. Данное обстоятельство настолько важно, что рассмотрим его отдельно.

**Условие цикла** Для начала анекдот:

> Программист не явился на работу. В конторе переполошились, решили его проведать. Звонили, звонили в дверь - никто не открывает. Из-за двери только плеск воды слышен. Взломали дверь, заходят и наблюдают такую картину: сидит программист в ванне, волос на голове почти не осталось, но он тем не менее судорожно намыливает голову шампунем.
> Читают инструкцию к шампуню:
>
> 1. намочить голову,
> 2. выдавить небольшое количество шампуня на руку,
> 3. намыливать 2-3 минуты,
> 4. смыть водой,
> 5. повторить.

Анекдот показывает важность правильно и корректного написания условия завершения.

Поэтому не забывай об условии завершения цикла. Убедись, что оно когда-нибудь да сработает, иначе твоя программа будет работать "бесконечно". Последнее означает, что если мы таких гарантий дать не можем, то может потребоваться просто ограничить сверху то количество раз, которое выполняется цикл. В итоге в нашем случае один из вариантов правильного условия звучит как: "пока волосы не будут скрипеть, но не более 2х раз последнее на случай, если шампунь "испортился".

**Команда** `break` Команда, позволяющая "досрочно"выйти из цикла. Так, обычно проверка условия проводится перед началом выполнения тела цикла. В случае успеха, выполняется цикл, иначе цикл больше не выполняется. Так вот, инструкция `break`, вызываемая внутри тела цикла, позволяет сразу, не дожидаясь проверки условия, выйти из цикла. Обычно используется для быстрого (аварийного) выхода из цикла.

Пример...

```
1  int s=0;//Значение суммы полагается равной 0.
2  while( 1 )
3  {
4          int a;
5          printf("enter a:");//приглашение на ввод
6          if( scanf("%d", &a) != 1 )//В случае ошибки
7                  break;//выходим из цикла (к стр. 11).
8          //В случае успешного считывания.
9          s = s + a;//Суммируем считываемые числа.
10 }
11 printf("sum ~ %d\n", s);
```

Можно переписать цикл и без использования break.

Упражнение. Перепиши цикл так, что условием цикла и будет факт успешного считывания. Лишнюю печать можно убрать.

Должно было получиться:

```
1  int s=0;//Значение суммы полагается равной 0.
2  int a;//Выносим переменную a перед блоком кода.
3  while( scanf("%d", &a) != 1 )
4  {
5          //В случае успешного считывания.
6          s = s + a;//Суммируем считываемые числа.
7  }
8  printf("sum ~ %d\n", s);
```

Упражнение. Найдите среднее значение вводимых чисел.

Последнее показывает, что без `break` можно обойтись, и иногда это упрощает код. Но бывает, что код наоборот усложняется.

Упражнение. Теперь восстанови вывод текста как он был в первоначальной программе. Вывод программы должен совпадать в точности. В чем неудобство?

Тем не менее, при правильном использовании break программа обычно упрощается.

**Команда `continue`** Команда `continue` (см. стр. 10), выполняемая в теле цикла, позволяет сразу выйти на проверку условия цикла (см. 2), т.е. на его самое начало. Обычно используется при проверке данных на некое условие (корректность), которое не влечет выхода из цикла.

```
1  int s=0;
2  while( 1 )
3  {
4          int a;
5          printf("enter a:");//приглашение на ввод
6          if( scanf("%d", &a) != 1 )
7                  break;//В случае ошибки выходим из цикла.
8          //Успешно считали некое целое число.
9          if( a < 0 )//В случае отрицательного числа
10                 continue;//заново к началу (к стр. 2).
11         //В случае неотрицательных чисел (a >= 0).
12         s = s + a;//Суммируем неотрицательные числа.
13 }
14 printf("summa ~ %d\n", s);
```

Можно обойтись и без continue.

```
1  int s=0;
2  while( 1 )
3  {
4          int a;
5          printf("enter a:");//приглашение на ввод
6          if( scanf("%d", &a) != 1 )
7                  break;//В случае ошибки выходим из цикла.
8          if( a >= 0 )//В случае неотрицательного числа
9          {
10                 s = s + a;//Суммируем числа.
11         }//else не требуется.
```

```
12 }
13 printf("summa ~ %d\n", s);
```

Видно, что программа "усложнилась"(текст сдвинулся в право).

Упражнение. Убери break ещё раз. Лишний вывод при необходимости тоже можно убрать.

## 1.2 Продвинутое понимание логических выражений

Рассмотрим следующий фрагмент кода:

```
1  int a;
2  printf("enter a:");
3  if( scanf("%d", &a) != 1 )
4  {
5          printf("error\n");
6          return -1;
7  }
8  //Считано число.
9  //Далее проверяем ещё одно условие.
10 if( a < 0 )
11 {
12         printf("error\n");
13         return -1;
14 }
```

Отмечу, что порядок команд и, в частности, проверок важен. Так, сначала необходимо попробовать считать число (см. стр. 3). В случае ошибки ввода программа завершает работу (см. стр. 6) предварительно выведя причину ошибки (см. стр. 5). Иначе число считывается успешно и программа продолжает работу со строчки 8.

По аналогии проверяется корректность считаного числа (то что оно неотрицательное). Важно то, что нельзя проверить в параллель, тем более, заранее корректность числа, т.е. код должен выполнятся строго последовательно.

Проблема в данном коде то, что он громоздкий: так не принято в идеологии языка Си. Мы могли бы условия конструкций if объединить в единое логическое выражение посредством операции ИЛИ и получить код:

```
1  int a;
2  printf("enter a:");
3  if( (scanf("%d", &a) != 1) || ( a < 0 ) )//Объединили
4  {                                        //условия.
```

```
5           printf("error\n");
6           return -1;
7   }
```

Оно истинно, если истинно хотя-бы одно из выражений. Но есть нюанс.

Рассмотрим случай, когда введена чепуха и, соответственно, `scanf` не удалось считать число и было возвращено 0е значение (т.е. истинно первое подвыражение). В таком случае переменной `a` не было ничего присвоено, и в лучшем случае она содержит мусор. Тогда вторая проверка ($a < 0$) некорректна, ведь переменной `a` не было ничего присвоено и поэтому, вообще говоря, мы не имеем право её использовать в выражениях (результат не будет детерминирован). Но, логические выражения, построенные на основе связок И и ИЛИ, на самом деле, обладают одним интересным свойством.

Напомню, что операция ИЛИ истинна тогда и только тогда, когда хотя бы один из операндов истинен. Поэтому, если какой-либо из них оказался истинным, то в общем-то нет смысла в проверке второго, так как он уже не способен изменить итоговый результат данной операции. Аналогично, если в связке И один из операндов ложен, то по тем же причинам, что и ранее, нет причин для проверки ложности второго (он уже в целом будет ложен).

Осталось вспомнить, что программа выполняется последовательно, т.е. она не может выявить истинность обоих операндов одновременно. Поэтому, необходимо установить четкий порядок, в котором будут проверяться операнды, а именно – они проверяются в порядке их появления в тексте программы (точнее в логическом выражении).

Исходя из выше написанного становится понятно, что ничего криминального не будет в случае, если переменная `a` не будет считана, так как в этом случае второе условие, в приведенном ранее фрагменте, просто не будет проверятся – для вычисления итоговой истинности выражения достаточно, что истинно перовое подвыражение.

В дальнейшем это будет использовано.

Пример с циклами!!!

## 1.3 Другие виды

Можно обойтись циклом while, но иногда для краткости или большей четкости, цикл можно записать через более частные конструкции. На самом деле они все взаимозаменяемые. Тот или иной цикл применяется исходя из вложенного в него смысла.

**Цикл for** Предполагает явное использование счетчиков, что фактически основное выражение цикла выполняет действия со значения a по значение b. Но это все

необязательно. Первый пример из циклов преобразуется так:

```
1  //for(инициализация; лог. выраж.; арифметич. действия )
2  //      команда;//естественно, может быть пустым.
3  //
4  int s, a;
5  printf("enter a:");//приглашение на ввод
6  scanf("%d", &a);//считываем значение
7  //Весь цикл сократится до одной строчки.
8  for(s=i=0; s < a; s = s + 1);
9  printf("summa ~ %d\n", s);
```

Инициализация переменных переносится в качестве первой группы внутри for конструкции (см. стр. 8). Выражение становится второй группой. Третья группа это простые арифметические действия, т.е. там, например, не может быть конструкции if, а присваивать переменным значения можно. В данном примере программа сократилась до одной строчки.

Программу можно записать иначе:

```
1  //for(инициализация; лог. выраж.; арифметич. действия )
2  //      команда;//теперь будет.
3  //
4  int s, a;
5  printf("enter a:");//приглашение на ввод
6  scanf("%d", &a);//считываем значение
7  //Весь цикл сократится до одной строчки.
8  for(s=i=0; s < a;)//арифмет. действия нет.
9           s = s + 1;//основная команда, вынесена.
10 printf("summa ~ %d\n", s);
```

Теперь мы тоже действие делаем явно в основном действии (см. 9).

Можно делать и больше одного действия:

```
1  int s, i, a;
2  printf("enter a:");
3  scanf("%d", &a);
4  //Опять весь цикл сократится до одной строчки.
5  for(s=i=0; s < a; i = i + 1, s = s + 1 );
6  printf("summa ~ %d\n", s);
```

Действия перечисляются через запятую, но if и других более сложных конструкций нет и быть не может.

Но можно было записать программу длиннее.

```
1 int s, i, a;
2 printf("enter a:");
3 scanf("%d", &a);
4 for(s=i=0; s < a; )
5 {
6         i = i + 1;
7         s = s + 1;
8 }
9 printf("summa ~ %d\n", s);
```

**do while**

## 1.4 Итого

**Ты знаешь**

**Эталон**

**упражнения**

1. *Фильтр введенных чисел* Найти количество/среднее введенных чисел удовлетворяющих условию, например, положительных четных.

2. *Сумма ряда* Найти сумму ряда до некого номер $k$ или найти наименьший номер $k$, что частичная сумма больше введенного числа.

3. *Рекурентные последовательности* Вывести $k$-ый член последовательности Фибоначи.