



ЯЗЫК ПАСКАЛЬ (**PASCAL**). ЦИКЛЫ

ОПЕРАТОРЫ ЦИКЛА

Циклической алгоритмической структурой (циклом) считается такая структура, в которой некоторые действия выполняются несколько раз. В программировании имеются два вида *циклических структур*: *цикл с параметром* и *итерационный цикл*.

В *цикле с параметром* всегда имеются так называемые *параметры цикла*: X , Xn , Xk , ΔX . Иногда *цикл с параметром* называют *регулярным циклом*. Характерной чертой является то, что число циклов и повторений можно определить до выполнения цикла.

В *итерационном цикле* невозможно определить число циклов до его выполнения. Он выполняется до тех пор, пока выполняется условие продолжение цикла.

В языке *Паскаль* имеются три оператора, реализующих циклические вычислительные структуры:

□ *счетный оператор* **for** (*оператор цикла с параметром*). Он предназначен для реализации *цикла с параметром* и не может быть использован для реализации *итерационного цикла*;

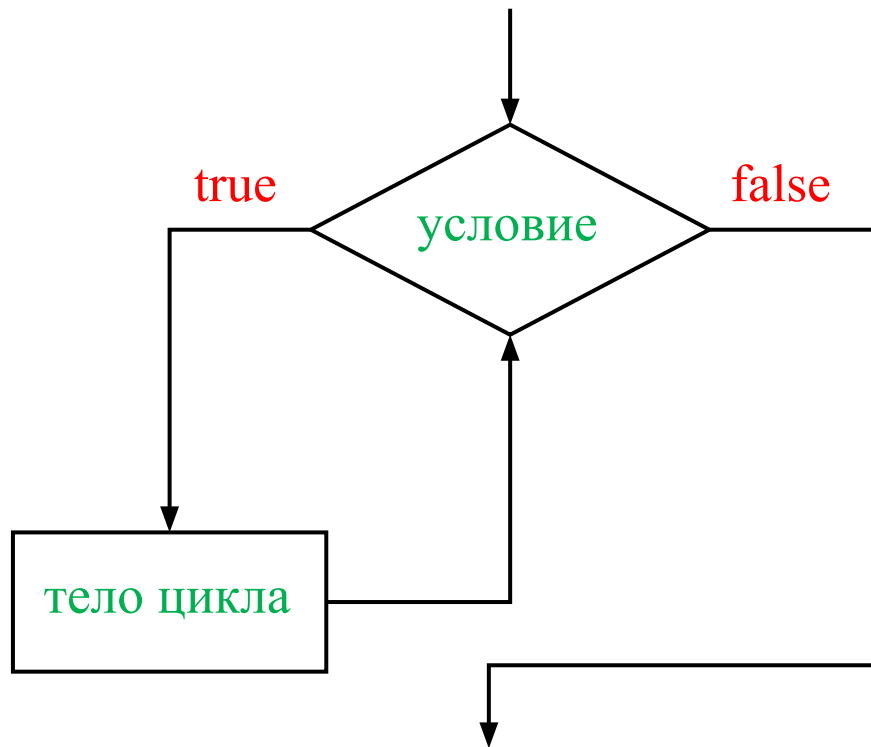
□ *оператор цикла с предусловием* **while** (*цикл-ПОКА*);

□ *оператор цикла с постусловием* **repeat** (*цикл-ДО*).

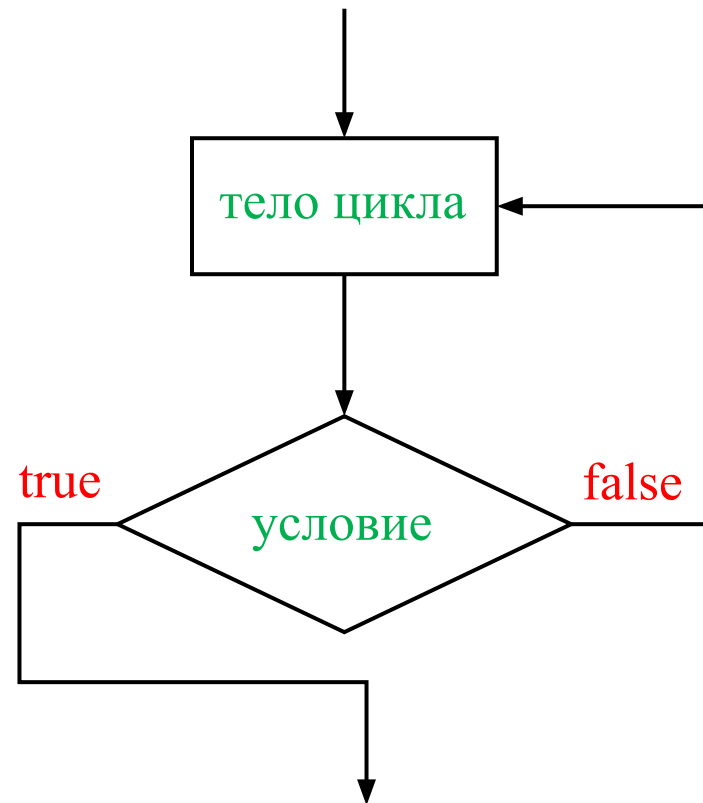
Последние два ориентированы на реализацию *итерационного цикла*, однако их можно использовать и для реализации *цикла с параметром*.

ОПЕРАТОРЫ ЦИКЛА

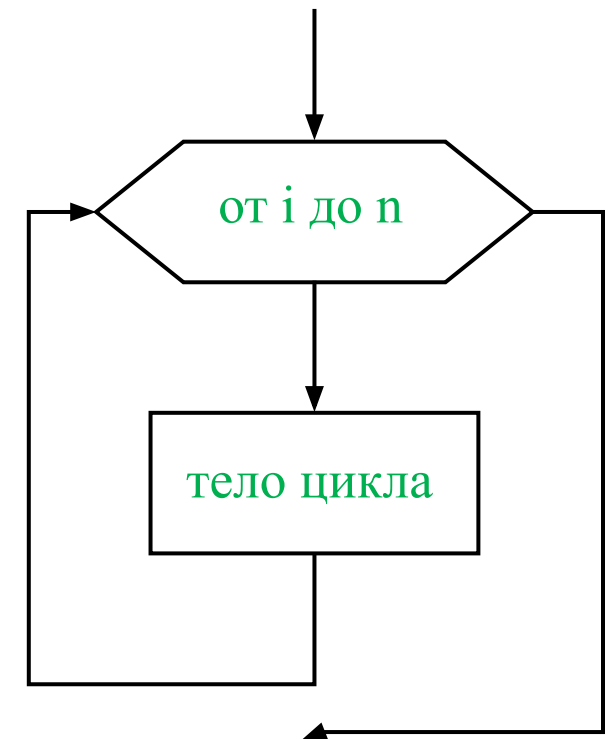
Цикл-ПОКА (**while**)



Цикл-ДО (**repeat**)



Цикл с параметром (**for**)



ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (FOR)

Если число повторений тела цикла заранее известно, то используется оператор цикла **for** (*оператор цикла с параметром*).

Оператор **for** состоит из двух частей: *тела цикла* и *заголовка*, который предназначен для описания начального и конечного значений параметра цикла, а также варианта его изменения (*возрастание* – **to** или *убывание* – **downto**):

```
{нач_знач <= кон_знач, т.е. параметр изменяется от меньшего значения к большему}  
for [параметр] := [нач_знач] to [кон_знач] do  
  [оператор]; {тело цикла}
```

```
{нач_знач >= кон_знач, т.е. параметр изменяется от большего значения к меньшему}  
for [параметр] := [нач_знач] downto [кон_знач] do  
  [оператор]; {тело цикла}
```

где **for** (*для*), **to/downto** (*увеличивать/уменьшать до*) и **do** (*выполнять, делать*) – служебные слова, [параметр] – переменная порядкового типа, называемая *параметром цикла*, [нач_знач] и [кон_знач] – выражения того же типа, что и *параметр цикла*, [оператор] – оператор, который выполняется многократно в цикле, называемый *телом цикла*.

ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (FOR)

Замечание: *Изменять* [параметр] в теле цикла *нельзя* – это вызовет ошибку.

Работа оператора цикла с параметром (for):

- [параметр] приравнивается [нач_знач] – минимальному (максимальному) возможному значению;
- проверяется, не превышает ли [параметр] [кон_знач], т.е. [параметр] \leq [кон_знач] (для **downto** – [параметр] \geq [кон_знач]). Если это условие выполняется, то идем на следующий пункт, иначе (т.е. при [параметр] $>$ [кон_знач]) – выходим из цикла (для **downto** – [параметр] $<$ [кон_знач]);
- выполняется [оператор] в теле цикла;
- далее увеличивается (уменьшается) [параметр] на 1, и цикл повторяется со 2 пункта (т.е. с проверки истинности условия [параметр] \leq [кон_знач], и т.д).

В роли начального значения [параметра] наиболее часто используют 1 или 0 (это зависит от задачи).

Когда начинает выполняться оператор **for**, **начальное** и **конечное значения определяются один раз**, и эти значения сохраняются на протяжении всего выполнения оператора **for**.

[Оператор], который содержится в теле оператора **for**, выполняется один раз для каждого значения в диапазоне между **начальным** и **конечным значением**.

Начальные и **конечные значения** могут быть представлены не только значениями, но и выражениями, возвращающими совместимые с типом параметра типы данных.

ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (FOR)

Оператор **for** существенно отличается от аналогичных операторов в других языках программирования. Отличия следующие:

- **тело оператора for**. Оператор может не выполняться ни разу, поскольку проверка условия продолжения цикла выполняется до тела цикла;
- **шаг изменения** параметра цикла постоянный и равен **1**;
- **тело цикла** в операторе **for** представлено одним оператором. В том случае, если действие тела цикла требует более одного простого оператора, то эти операторы необходимо превратить в один **составной оператор** посредством **операторных скобок** (**begin - end**). При этом, сами **[операторы]** разделяются оператором «**;**» ("точка с запятой", но перед закрывающим **end**, в конце, ставить её не обязательно);
- **параметр цикла** может быть только переменной порядкового типа.

Замечание: Считается, что при нормальном завершении выполнения **оператора цикла** в языке **Паскаль** значение **параметра цикла** не определено, но если цикл завершится раньше положенного, то **[параметр]** сохранит последнее, записанное в него значение.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА FOR

{Вывести на экран N символов «*», где N задает пользователь}

```
program progr_symbol;  {заголовок программы}
var  {раздел описания переменных}
  i, n: integer;  {объявляем 2 переменные целого типа}
begin  {начало тела программы}
  write ('Количество знаков: ');  {вывод сообщения}
  readln (n);  {ввод количества символов}
  for i := 1 to n do  {будем последовательно перебирать числа от 1 до N}
    write ('* ');  {вывод символов на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
    программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА FOR

{Посчитать сумму первых 1000 натуральных чисел}

```
program progr_sum1000;  {заголовок программы}
var  {раздел описания переменных}
  i, sum: integer;  {объявляем 2 переменные целого типа}
begin  {начало тела программы}
  sum := 0;  {начальное значение суммы}
  for i := 1 to 1000 do  {будем последовательно перебирать числа от 1 до 1000}
    sum := sum + i;  {к сумме прибавляем i}
  writeln ('Результат: ', sum);  {вывод результата на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
  программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```


ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА FOR

{Напечатать таблицу умножения на 7}

```
program progr_tab7;  {заголовок программы}
var  {раздел описания переменных}
  i: byte;  {объявляем переменную типа «байт»}
begin  {начало тела программы}
  for i := 1 to 10 do  {будем последовательно перебирать числа от 1 до 10}
    writeln ('7 × ', i, ' = ', 7 * i);  {вывод таблицы умножения на 7 на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
    программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА FOR

{Рассчитать значения «у» для значений «х», равных 4, 5, ..., 28, где
 $y = 2t^2 + 5.5t - 2$, если $t = x + 2$ }

```
program progr_formula;    {заголовок программы}
var    {раздел описания переменных}
  x, t: integer;    {объявляем 2 переменные целого типа}
  y: real;    {объявляем переменную вещественного типа}
begin    {начало тела программы}
  for x := 4 to 28 do    {будем последовательно перебирать числа от 4 до 28}
  begin    {операторные скобки - начало}
    t := x + 2;    {вычисляем «t»}
    y := 2 * t * t + 5.5 * t - 2;    {вычисляем «у»}
    writeln (' x = ', x:2, ', y = ', y)    {вывод результата на экран}
  end;    {операторные скобки - конец}
end.    {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА FOR (УБЫВАНИЕ – DOWNTO)

{Вычислить сумму первых 1000000 членов гармонического ряда}

```
program progr_garm_ryad;  {заголовок программы}
var  {раздел описания переменных}
  i: integer;  {объявляем 2 переменные целого типа}
  sum: real;  {объявляем переменную вещественного типа}
begin  {начало тела программы}
  sum := 0;  {начальное значение суммы}
  for i := 1000000 downto 1 do  {будем последовательно перебирать числа от 1000000 до 1}
    sum := sum + 1 / i;  {к сумме прибавляем 1/i – очередной член гармонического ряда}
  writeln ('Результат: ', sum);  {вывод результата на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
    программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ (WHILE)

Оператор цикла **while** используется тогда, когда заранее неизвестно общее количество *итераций* (повторений вычислений) цикла, а завершение вычислений зависит от некоторого условия, которое проверяется в начале цикла (на входе).

```
while [условие] do  
    [оператор];    {тело цикла}
```

где **while** (пока), **do** (делать, выполнять) – служебные слова,
[условие] – выражение логического типа,
[оператор] – обыкновенный оператор (выполняется 0 или более раз)

Замечание: *Оператор цикла Паскаля с предусловием* можно считать наиболее универсальным – с использованием таких операторов можно задать и циклические процессы, определяемые *операторами цикла с параметром и постусловием*.

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ (WHILE)

Оператор **while** работает следующим образом: в начале работы проверяется результат логического условия. Если результат **истина** (**true**), то выполняется оператор, после которого осуществляется возврат на проверку условия с новым значением параметров в логическом выражении условия. Если результат **ложь** (**false**), то осуществляется завершение цикла (если [**условие**] принимает значение **ложь** при первом же его вычислении, то [**оператор**] не выполнится ни разу).

Поэтому очень важно в теле цикла предусмотреть изменение переменной, фигурирующей в заголовке цикла, таким образом, чтобы когда-нибудь обязательно наступала ситуация **false**. Иначе произойдет **защелкивание** (*бесконечное выполнение программы или бесконечный цикл*), одна из самых неприятных ошибок в программировании.

При работе с **while** надо обратить внимание на *его свойства*:

- *условия*, использованные в **while**, являются *условием продолжения цикла*;
- в *теле цикла* всегда происходит *изменение значения параметра* входящего в выражение условия;
- цикл **while** может, *не выполниться ни разу*, поскольку проверка условия в продолжение цикла выполняется до тела цикла.

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ (WHILE)

Если [оператор] в цикле **while** состоит из нескольких операторов, то поместить их нужно в операторные скобки (**begin** - **end**), а сами операторы нужно разделять оператором «**;**» ("точка с запятой", но перед закрывающим **end**, в конце, ставить её не обязательно).

```
while [условие] do  
begin  
    [оператор 1];  
    [оператор 2];  
    [оператор 3];  
    .....  
    [оператор N]  
end;
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА WHILE

{Напечатать минимальное число, большее 200, которое нацело делится на 17}

```
program progr_min200;  {заголовок программы}
var  {раздел описания переменных}
  n: integer;  {объявляем переменную целого типа}
begin  {начало тела программы}
  n := 201;  {минимальное число, большее 200}
  while (n mod 17 <> 0) do  {будем выполнять цикл, пока введенное число не делится нацело}
    inc (n);  {увеличиваем значение делителя «d» на 1}
  writeln ('Ответ: ', n);  {вывод результата на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
  программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА WHILE

{Выведите наименьший делитель числа x, отличный от 1}

```
program progr_min_delitel;  {заголовок программы}
var  {раздел описания переменных}
  x, d: integer;  {объявляем 2 переменные целого типа}
begin  {начало тела программы}
  write ('ВВЕДИТЕ x --> ');  {вывод сообщения}
  readln (x);  {ввод числа}
  d := 2;  {минимальный делитель отличный от 1}
  while (x mod d <> 0) do  {будем выполнять цикл, пока введенное число не делится нацело}
    inc (d);  {увеличиваем значение делителя «d» на 1}
  writeln ('d = ', d);  {вывод результата на экран}
  readln  {данный оператор может отсутствовать, применяется в Turbo Pascal, чтобы
    программа сразу не закрывалась после её выполнения}
end.  {конец тела программы}
```


ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА WHILE

{Вычисление суммы кубов всех чисел от 1 до 10}

```
program progr_kubi10;  {заголовок программы}
var  {раздел описания переменных}
  b, sum: integer;  {объявляем 2 переменные целого типа}
begin  {начало тела программы}
  b := 1;  {начальное значение переменной «b»}
  sum := 0;  {начальное значение суммы}
  while b <= 10 do  {будем выполнять цикл, пока значение переменной «b» <= 10}
  begin  {операторные скобки - начало}
    sum := sum + b * b * b;  {к сумме прибавляем куб очередного числа}
    inc (b)  {увеличиваем значение переменной «b» на 1}
  end;  {операторные скобки - конец}
  writeln ('sum = ', sum);  {вывод результата на экран}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА WHILE

{Разложение функции e^x в ряд Тейлора (с точностью 0.000001 (одна миллионная))}

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, \quad -\infty < x < \infty$$

```
program progr_taylor;  {заголовок программы}
var a, x, s: real;     {раздел описания переменных, 3 переменные вещественного типа}
    n: integer;       {объявляем переменную целого типа}
begin  {начало тела программы}
    x := 0.5; n := 0; a := 1; s := 0;
    {n - начальный номер, a - начальное значение члена ряда, s - начальная сумма}
    while a > 0.000001 do  {будем выполнять цикл, пока значение переменной «a» > 0.000001}
    begin  {операторные скобки - начало}
        s := s + a;  {к сумме прибавляем очередной член ряда Тейлора}
        inc (n);    {увеличиваем значение начального номера «n» на 1}
        a := a * x / n  {вычисляем значение очередного члена ряда Тейлора}
    end;  {операторные скобки - конец}
    writeln ('s = ', s:0:6);  {вывод результата на экран}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА WHILE

{На вход программе поступает натуральное число. Определите количество разрядов в числе}

{Фрагмент кода программы}

readln (num); *{ввод числа}*

k := 0; *{начальное значение количества разрядов в числе}*

while (num <> 0) **do** *{будем выполнять цикл, пока значение переменной «num» не равно 0}*

begin *{операторные скобки - начало}*

inc (k); *{увеличиваем значение счетчика «k» на 1}*

 num := num **div** 10; *{делим значение переменной «num» на 10 нацело}*

end; *{операторные скобки - конец}*

writeln (k); *{вывод результата на экран}*

ПРИМЕР «БЕСКОНЕЧНОГО ЦИКЛА»

{Рассмотрим простейший пример – вывод на экран сообщения «Hello!». В данном примере условие входа в цикл всегда истинно. Такая ситуация называется "бесконечным циклом"}

{Фрагмент кода программы}

Begin *{начало тела программы}*

while $1 < 100$ **do** *{будем выполнять цикл, пока $1 < 100$, т.е. бесконечное число раз}*

writeln ('Hello!') *{вывод на экран сообщения «Hello!»}*

end. *{конец тела программы}*

ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ (REPEAT)

Оператор цикла **repeat** используются тогда, когда заранее не известно общее количество *итераций* (повторений вычислений) цикла, а завершение вычислений зависит от некоторого условия, которое проверяется в конце цикла (на выходе).

repeat

[оператор] {тело цикла}

until [условие];

где **repeat** (повторять) и **until** (до) – служебные слова,

[оператор] – любой оператор *Паскаля* (выполняется 1 или более раз),

[условие] – логическое выражение.

Замечание: Любой *оператор с предусловием* **while** легко может быть преобразован в *оператор с постусловием* **repeat**, т.к. хотя бы 1 раз оператор в теле цикла выполнится, поскольку условие выхода проверяется в конце. Но далеко не каждый оператор **repeat** легко записывается с помощью **while**.

ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ (REPEAT)

Оператор **repeat** работает следующим образом: сначала выполняются *операторы тела цикла* (между ключевыми словами **repeat** и **until**), после чего проверяется результат логического условия. Если результат **ложь** (**false**), то осуществляется возврат к выполнению *операторов очередного тела цикла*. Если результат **истина** (**true**), то оператор завершает работу (происходит выход из цикла).

Оператор **repeat** имеет следующие особенности:

- в **repeat** проверяется условие завершения цикла и если *условие выполняется*, то *цикл прекращает работу*;
- *тело цикла* всегда *выполняется* хотя бы *один раз*;
- *параметр* для проверки условия *изменяется* в теле цикла;
- *операторы тела цикла* не надо заключать в операторские скобки (**begin-end**), при этом роль операторных скобок выполняют **repeat** и **until**.

Замечание: *Оператор цикла с постусловием* является *более общим*, чем *оператор цикла с параметром* – любой циклический процесс, задаваемый с помощью *цикла с параметром* можно представить в виде *цикла с постусловием*. *Обратное утверждение неверно.*

ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ (REPEAT)

Отличие оператора цикла **repeat** от **while** состоит в том, что в нем условие проверяется на выходе из цикла: если оно не выполняется, то цикл продолжается, если выполнится – сразу выход из цикла, т. е. пока условие **истинно** (**true**), программа идет на следующую итерацию, условие нарушается (**false**) – выходим. Поэтому оператор **repeat** ещё называют *оператором выхода*. Ещё в операторе **repeat** *не нужны операторные скобки* (**begin-end**) для нескольких операторов:

repeat

[оператор 1];

[оператор 2];

[оператор 3];

.....

[оператор N]

until [условие];

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Задача о суммировании первых n членов гармонического ряда (ранее рассмотрена на примере цикла **for**)}

{Фрагмент кода программы}

readln (n); *{ввод количества членов гармонического ряда}*

$i := 0$; *{начальное значение номера члена гармонического ряда}*

$y := 0$; *{начальное значение суммы}*

repeat *{начало цикла repeat}*

$i := i + 1$; *{переходим к очередному члену ряда}*

$y := y + 1 / i$; *{к сумме прибавляем очередной член гармонического ряда}*

until $i > n$; *{будем выполнять цикл, пока значение переменной «i» не превысит «n»}*

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Вычислить корень квадратный из введенного с клавиатуры числа}

```
program progr_square_root;  {заголовок программы}
var  {раздел описания переменных}
  x: integer;  {объявляем переменную целого типа}
begin  {начало тела программы}
  repeat  {начало цикла repeat}
    readln (x)  {ввод числа}
  until x >= 0;  {выходим из цикла, если x >= 0}
  writeln ('Квадратный корень: ', sqrt (x):0:4);  {вывод результата на экран}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Задача. Вводить с клавиатуры числа до тех пор, пока их сумма не превысит заданное наперед число}

```
program progr_verh_gr;  {заголовок программы}
var  {раздел описания переменных}
  x, m, sum: real;  {объявляем 3 переменные вещественного типа}
begin  {начало тела программы}
  write ('Введите контрольное число --> ');  {вывод сообщения}
  readln (m);  {ввод числа}
  sum := 0;  {начальное значение суммы}
  repeat  {начало цикла repeat}
    readln (x);  {вводим x}
    sum := sum + x  {к сумме прибавляем x}
  until sum > m;  {выходим, если сумма превысит m}
  writeln ('Результат: ', sum);  {вывод результата на экран}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Вводится последовательность чисел, 0 – конец последовательности. Определить, содержит ли последовательность хотя бы два равных соседних числа}

```
program progr_square;  {заголовок программы}
var  {раздел описания переменных}
  n, pre: integer;  {объявляем 2 переменные целого типа}
  twoEqual: boolean;  {объявляем переменную логического типа}
begin  {начало тела программы}
  twoEqual := false;  {равных чисел нет (на данный момент)}
  pre := 0;  {начальное значение предыдущего числа}
  writeln ('Введите элементы последовательности: ');  {вывод сообщения}
  repeat  {начало цикла repeat}
    read (n);  {ввод числа}
    {Вычисление twoEqual имеет смысл, если мы ещё не встретили два равных соседних числа (т.е. при twoEqual=false). Если последнее введенное число n равно предыдущему pre (pre=n), то индикатор twoEqual станет истинным и больше не изменится}
    if not twoEqual then twoEqual := (n = pre);
    pre := n  {предыдущим pre становится n}
  until n = 0;  {выходим при вводе 0}
  writeln;
  if twoEqual then writeln ('Содержит')  {вывод результата на экран}
  else writeln ('Не содержит');
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Вводится последовательность из N целых чисел. Найти наибольшее из всех отрицательных чисел}
{Первый вариант решения}

```
program progr_max_negative_1;  {заголовок программы}
var n, i, a, maxNegNum: integer;  {раздел описания переменных, объявляем 4 переменные целого типа}
begin  {начало тела программы}
  i := 0;  {номера вводимых чисел}
  n := 10;  {количество вводимых чисел}
  maxNegNum := -MaxInt - 1;  {начальное значение максимального отрицательного числа}
  repeat  {начало цикла repeat}
    inc (i);  {увеличиваем номер вводимого числа }
    read (a);  {вводим число}
    {Для поиска максимального числа среди отрицательных элементов добавим условие  $a < 0$ . При каждой итерации
    сравниваем введенное число с максимальным, если оно больше максимального, то заменим им максимальное}
    if (a < 0) and (a > maxNegNum) then
      maxNegNum := a
  until i = n;  {выходим из цикла, если введены все n чисел}
  writeln ('Ответ: ', maxNegNum);  {выводим результат}
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Вводится последовательность из N целых чисел. Найти наибольшее из всех отрицательных чисел}
{Второй вариант решения}

{Фрагмент кода программы}

begin *{начало тела программы}*

i := 0; {количество введенных чисел}

n := 10; {количество чисел для ввода}

maxNegNum := 0; {максимальное значение}

writeln ('Введите ', n, ' целых чисел:'); *{вывод сообщения}*

repeat *{начало цикла repeat}*

inc (i); **write** (i, ' '); **readln** (a); *{увеличиваем номер вводимого числа и вводим само число}*

if (a < 0) **then** *{проверяем только отрицательные числа}*

{Когда в наборе мы нашли первое отрицательное число, величина maxNegNum равна 0, поэтому меняем её на a – это и будет первое ненулевое значение для maxNegNum}

if maxNegNum = 0 **then** maxNegNum := a

else *{в остальных случаях maxNegNum сравниваем с a и находим большее}*

if (a > maxNegNum) **then** maxNegNum := a

until i = n; *{выходим, когда введены все числа}*

{Когда переменная maxNegNum отрицательная, то это означает, что в наборе чисел есть отрицательные – тогда выводим максимальное из них, в противном случае сообщаем об отсутствии отрицательных чисел}

if maxNegNum < 0 **then** **writeln** ('Максимальное отрицательное число: ', maxNegNum) *{выводим результат}*

else **writeln** ('В последовательности нет отрицательных чисел');

ПРИМЕР ИСПОЛЬЗОВАНИЯ ЦИКЛА REPEAT

{Вводится последовательность из N целых чисел. Найти наибольшее из всех отрицательных чисел}
{Третий вариант решения}

```
program progr_max_negative_3;  {заголовок программы}
var n, i, a, maxNegNum: integer;  {раздел описания переменных, объявляем 4 переменные целого типа}
    bln: boolean;  {объявляем переменную логического типа}
begin  {начало тела программы}
    i := 0; n := 10; bln := false;
    writeln ('Введите ', n, ' целых чисел:');  {вывод сообщения}
    repeat  {начало цикла repeat}
        inc (i); write (i, ' '); readln (a);  {увеличиваем номер вводимого числа и вводим само число}
        if (a < 0) then  {проверяем только отрицательные числа}
            if not bln then begin
                maxNegNum := a;
                bln := true
            end
        else
            if (a > maxNegNum) then maxNegNum := a
    until i = n;  {выходим, когда введены все числа}
    if not bln then writeln ('Нет отрицательных чисел')  {выводим результат}
    else writeln ('Ответ: ', maxNegNum);
end.  {конец тела программы}
```

ОТЛИЧИЯ И ОСОБЕННОСТИ ХОРОШЕГО СТИЛЯ РАБОТЫ С ЦИКЛАМИ

Цикл с предусловием WHILE (пока условие истинно)

Цикл с постусловием REPEAT (до истинности условия)

До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл

В теле цикла **должны присутствовать операторы**, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился

Цикл работает пока условие **истинно** (пока **True**)

Цикл работает пока условие **ложно** (пока **False**)

Цикл завершается, когда условие становится **ложным** (до **False**)

Цикл завершается, когда условие становится **истинным** (до **True**)

Цикл может не выполняться ни разу, если исходное значение условия при входе в цикл **False**

Цикл обязательно выполнится как минимум один раз

Если в теле цикла требуется выполнить более одного оператора, то **необходимо использовать составной оператор**

Независимо от количества операторов в теле цикла, **использование составного оператора не требуется**

Цикл со счетчиком (с параметром) FOR

Начальная установка переменной счетчика цикла до заголовка не требуется

Изменение в теле цикла **значений переменных**, стоящих **в заголовке не допускается**

Количество итераций цикла неизменно и точно определяется значениями **нижней** и **верхней границ** и **шага приращения**

Нормальный ход работы цикла может быть нарушен оператором **goto** или процедурами **break** и **continue**

Цикл может не выполняться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе

ВЛОЖЕННЫЕ ЦИКЛЫ

Оператор, который выполняется в *цикле*, сам может быть *циклом*. Это относится ко всем видам *циклов*. В результате мы получаем *вложенные циклы*.

Замечание: *Главным обстоятельством при работе со вложенными циклами является использование разных переменных для счетчиков внутреннего и внешнего циклов*

Рассмотрим механизм работы *вложенных циклов* на примере работы электронных часов, начиная с момента времени 0 часов, 0 минут, 0 секунд.

Значение «минут» станет равным 1 только после того, как секунды «пробегут» все последовательные значения от 0 до 59. «Часы» изменят свое значение на 1 только после того, как минуты «пробегут» все последовательные значения от 0 до 59. Для вывода всех значений времени от начала суток и до конца можно воспользоваться следующим фрагментом программы:

```
for h := 0 to 23 do  
for m := 0 to 59 do  
for s := 0 to 59 do  
writeln (h, ':', m, ':', s);
```


ПРИМЕР ИСПОЛЬЗОВАНИЯ ВЛОЖЕННЫХ ЦИКЛОВ

{Вывести таблицу умножения, используя вложенные циклы в Паскале}

```
program progr_tab_multi;  {заголовок программы}
const n = 9;  {размер таблицы}
var i, j: integer;
begin  {начало тела программы}
  for i := 1 to n do  {номера строк}
  begin  {операторные скобки - начало}
    for j := 1 to n do  {номера столбцов}
      write (i * j : 4);
    writeln;  {переход на новую строку}
  end;  {операторные скобки - конец}
end.  {конец тела программы}
```

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА GOTO

goto [метка];

goto – зарезервированное слово в языке *Паскаль*. [метка] – это произвольный *идентификатор*, который позволяет пометить некий оператор программы и в дальнейшем сослаться на него. В языке *Паскаль* допускается в качестве *меток* использовать целое число без знаков. *Метка* располагается перед помеченным оператором и отделяется от него («:»). Один оператор можно помечать несколькими *метками*. Они так же отделяются друг от друга («:»). При выполнении программы оператор **goto** осуществляет переход к помеченному оператору программы (перед ним стоит *метка*).

Пример: {Фрагмент кода программы}

```
label 1;  
begin  
    ....  
    goto 1;  
    ....  
    1: writeln ('Переход к метке 1');  
end.
```

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА GOTO

При использовании *меток* нужно руководствоваться следующими *правилами*:

- *метка* должна быть описана в разделе **label** (раздел описания) и все *метки* должны быть использованы;
- если в качестве *меток* используются целые числа, их не объявляют.

Пример: **label** lb1, lb2, met1;

```
.....  
begin  
.....  
lb1: [оператор];  
goto met1;  
.....  
lb2: [оператор];  
.....  
goto lb1;  
met1: [оператор];  
end.
```

!!!Замечание: Оператор **goto** *противоречит принципам технологии структурного программирования. Современные языки программирования его не имеют, и нет необходимости в использовании goto.* В современных компьютерах используется так называемый *конвейерный способ*. Если в программе встречается оператор *безусловного перехода*, то такой оператор ломает весь конвейер, заставляя создавать его заново, что существенно замедляет вычислительный процесс.

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА GOTO

{Вычисление квадратного корня}

```
program progr_sqrt_goto;  {заголовок программы}  
label goback;  {раздел описания меток}  
var num: real;  {раздел описания переменных}  
Begin  {начало тела программы}  
  goback:  {метка}  
  write ('Введите число: ');  {вывод сообщения}  
  readln (num);  {ввод числа}  
  if num < 0 then  {проверка условия}  
    goto goback;  {переход на метку}  
  num := sqrt (num);  {вычисление квадратного корня}  
  write ('Квадратный корень: ', num:5:2);  {вывод результата на экран}  
end.  {конец тела программы}
```

ОПЕРАТОРЫ BREAK, CONTINUE И EXIT

При использовании *циклов* в своей программе может понадобиться досрочно выйти из *цикла* или пропустить несколько последних операторов в *цикле*. Для этого используются операторы **break** и **continue**. При необходимости досрочно выйти из *процедуры* или *функции* или вообще завершить программу используется оператор **exit**.

Замечание: операторы **break** и **continue** применяются только внутри циклов (**repeat**, **while**, **for**) и действительны только для внутреннего цикла (если нужно обеспечить принудительный выход из двойного цикла, оператор **break** должен быть расположен как во внутреннем, так и во внешнем цикле). Операторы **break** и **continue** по сути являются видоизмененными операторами **goto** с известной точкой, в которую осуществляется переход. Оператор **break** выполняет полный выход из цикла, т.е. все возможные итерации цикла прерываются. Оператор **continue** прерывает только текущую итерацию и переходит к проверке условия продолжения (прекращения) цикла. **Break** и **continue** являются процедурами, хотя обычно их называют операторами. **Exit** – это оператор, предназначенный для досрочного выхода из процедуры или функции и возвращения в основную программу. Вызов оператора **exit** в основной программе приводит к её завершению, т.е. его можно применять и вне цикла (в отличие от **break** и **continue**).

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА BREAK

{Напечатаем все символы английского алфавита от "a" до "m"}

```
program progr_symb_am;    {заголовок программы}
var    {раздел описания переменных}
  ch: char;    {переменная ch символьного типа }
begin    {начало тела программы}
  {перебираем все символы английского алфавита}
  for ch := 'a' to 'z' do
  begin
    write (ch : 2);    {выводим символ}
    {когда встречаем символ 'm', выходим из цикла: }
    if ch = 'm' then
      break
  end;
end.    {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА BREAK

{Будем выводить случайные целые числа от 0 до 99, пока не встретится число, большее 90. И так 10 раз с новой строки}

```
program progr_099;  {заголовок программы}
var  {раздел описания переменных}
  i, n: integer;  {2 переменные целого типа}
begin  {начало тела программы}
  randomize;  {«включаем» генератор случайных чисел}
  for i := 1 to 10 do begin  {повторяем 10 раз}
    while true do begin
      n := random (100);  {случайное число 0..99}
      write (n : 3);  {выводим число}
      if n > 90 then  {если число больше 90 – выходим из цикла}
        break
    end;
    writeln  {переходим на следующую строку}
  end;
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА CONTINUE

{Из 10 случайных чисел с диапазона [-100, 99] вывести только положительные}

```
program progr_positive_10099;  {заголовок программы}
var  {раздел описания переменных}
  i, n: integer;  {2 переменные целого типа}
begin  {начало тела программы}
  randomize;  {«включаем» генератор случайных чисел}
  for i := 1 to 10 do  {повторяем 10 раз}
  begin
    n := -100 + random (200);  {диапазон чисел - [-100, 99]}
    if n < 0 then  {если число меньше 0 – переходим к следующей итерации}
      continue;
    write (' ', n)  {выводим число}
  end;
end.  {конец тела программы}
```


ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА CONTINUE

{Для 10 случайных чисел из интервала (-10, 10) вычислить их квадратные корни}

```
program progr_sqrt_10;  {заголовок программы}
var i: integer; a: real;  {раздел описания переменных, 1 переменная целого типа и 1 вещественного}
begin  {начало тела программы}
  i := 0;
  writeln ('| число | корень |');  {выводим сообщение}
  writeln ('-----');  {выводим сообщение}
  while i < 10 do begin  {будем выполнять цикл, пока значение переменной «i» < 10}
    inc (i);  {увеличиваем значение переменной «i» на 1}
    a := -10 + 20 * random;
    if a < 0 then begin  {если полученное число отрицательное, то корень вычислить нельзя}
      writeln ('|', a : 5 : 3, '|', '|': 9);
      continue  {если число меньше 0 – переходим к следующей итерации}
    end;
    writeln ('|', a : 5 : 3, '|', '|', sqrt (a) : 6 : 5, '|')  {вывод результата}
  end;
end.  {конец тела программы}
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА EXIT

{Вычислить квадратный корень из введенного пользователем числа}

```
program progr_sqrt;    {заголовок программы}
var    {раздел описания переменных}
  x: real;    {переменная вещественного типа}
begin    {начало тела программы}
  write ('Введите число --> ');    {выводим сообщение}
  readln (x);    {ввод числа}
  if x < 0 then    {если введенное число отрицательное, то корень вычислить нельзя}
  begin
    writeln ('ERROR!');
    exit    {завершаем работу программы если, если введено отрицательное число}
  end;
  writeln ('Корень квадратный = ', sqrt (x) : 0 : 5);    {вывод результата}
end.    {конец тела программы}
```

ПРИНУДИТЕЛЬНОЕ ПРЕКРАЩЕНИЕ ПРОГРАММЫ

Обычно программа завершает свою работу по достижении последнего оператора (т.е. при выходе на оператор **end** с точкой). Если возникает необходимость *прекратить выполнение программы* где-либо внутри нее, то можно воспользоваться *процедурой halt*, которая вызывается как отдельный оператор. Эту *процедуру можно вызвать*, задав в круглых скобках *параметр в виде целого неотрицательного числа* от **0** до **255**. Это значение возвращается в операционную систему в виде *кода ошибки (ERRORLEVEL)* и может быть проанализирована DOS в случае запуска данной программы из командного файла. Отсутствие параметра в процедуре **halt** соответствует значению параметра **0** (*нормальное завершение программы*).

Второй *процедурой*, с помощью которой можно *прекратить выполнение программы*, является *процедура без параметров exit* (при ее размещении в исполнимой части программы, а не в теле *подпрограммы*). Чаще эта процедура применяется для выхода из *подпрограммы* без прекращения выполнения вызывающей программы.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели *операторы цикла* языка *Pascal* (*for*, *while*, *repeat*), позволяющие некоторые действия в программе выполнять несколько раз. В программировании имеются два вида циклических структур: *цикл с параметром* и *итерационный цикл*. Для каждого вида приведено описание и примеры использования в программах.

В отдельную таблицу сведены отличия и особенности хорошего стиля работы с *циклами*. Рассмотрены на примерах оператор безусловного перехода (*goto*), оператор досрочного выхода из *цикла* (*break*), оператор досрочного выхода из текущей итерации *цикла* (*continue*), операторы для принудительного прекращения программы (*exit*, *halt*).

По ходу изложения материала было рассмотрено несколько примеров программ с применением различных операторов, реализующих *циклы*. Каждая строчка кода дополнена комментарием.