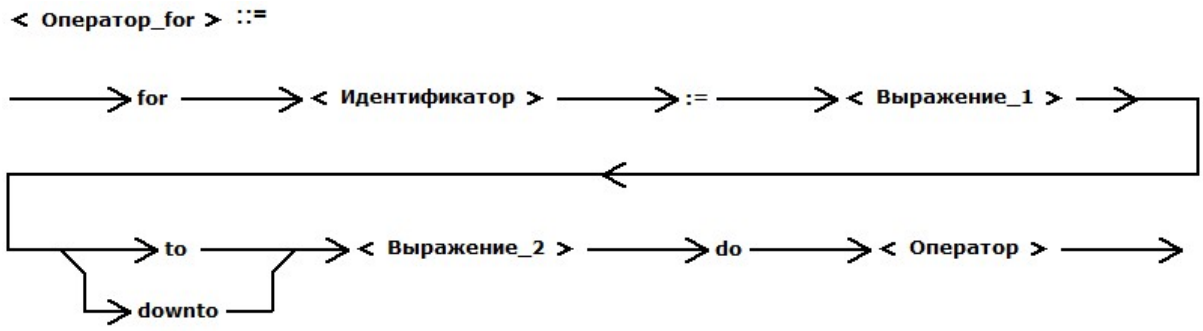
ТУБ №26:
Операторы организации циклов в языке Delphi

For — вкусен и скор

Практически любая программа не обходится без циклов: их содержат в себе даже сравнительно несложные алгоритмы, а уж если речь идёт о программе с графическим пользовательским интерфейсом, то циклы там есть и далеко не в единственном числе.

Часто в программе требуется повторить какие-то действия строго определённое число раз, да ещё и с каким-никаким подсчётом. На этот случай в большинстве языков программирования высокого уровня имеется оператор цикла с параметром for, и Delphi здесь не исключение:



В качестве тела цикла выступает единственный оператор, записываемый после ключевого слова do. В качестве параметра цикла выступает переменная, идентификатор которой записывается сразу после ключевого слова for. Переменная должна быть одного из перенумерованных типов. На каждой итерации эта переменная будет принимать новое значение из диапазона, заданного двумя выражениями после знака присваивания: если использовано ключевое слово to, то перебор значений будет осуществляться по возрастанию, иначе — по убыванию.

Запишем простой пример:

```
for I := 1 to 10 do
  WriteLn(I:4, ' ^ 2 = ', (I * I):8);
```

В данном случае телом цикла выступает WriteLn, причём выполнится он 10 раз, а переменная `I` будет принимать значения 1, 2, 3, …, 9, 10. Вывод программы:

```
   1 ^ 2 =     1
   2 ^ 2 =     4
   3 ^ 2 =     9
   4 ^ 2 =    16
   5 ^ 2 =    25
   6 ^ 2 =    36
   7 ^ 2 =    49
   8 ^ 2 =    64
   9 ^ 2 =    81
  10 ^ 2 =   100
```

Попробуем теперь в обратную сторону:

```
for I := 10 downto 1 do
  WriteLn(I:4, ' ^ 2 = ', (I * I):8);
```

Вывод программы:

```
 10 ^ 2 =  100
  9 ^ 2 =   81
  8 ^ 2 =   64
  7 ^ 2 =   49
  6 ^ 2 =   36
  5 ^ 2 =   25
  4 ^ 2 =   16
  3 ^ 2 =    9
  2 ^ 2 =    4
  1 ^ 2 =    1
```

Но к перенумерованным типам относятся не только целочисленные. Как насчёт символьного?

```
WriteLn('English alphabet:');
for C := 'A' to 'Z' do
  Write(C:2);
```

Вывод программы:

```
English alphabet:
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

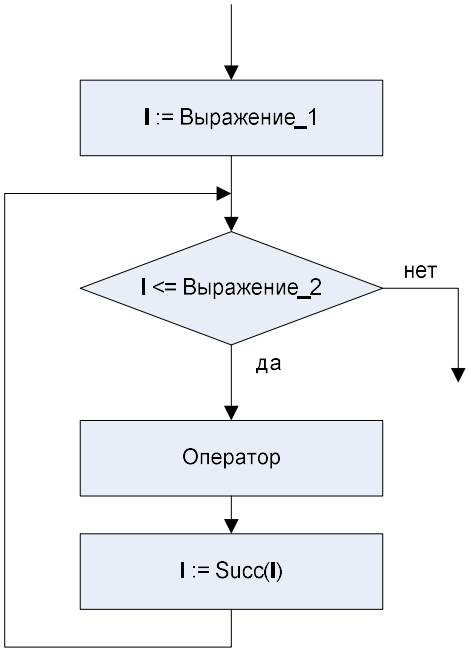
Работает это и с логическим типом, хотя, справедливости ради, это не то чтобы было очень уж полезно:

```
for B := False to True do
  WriteLn(B:6);
```

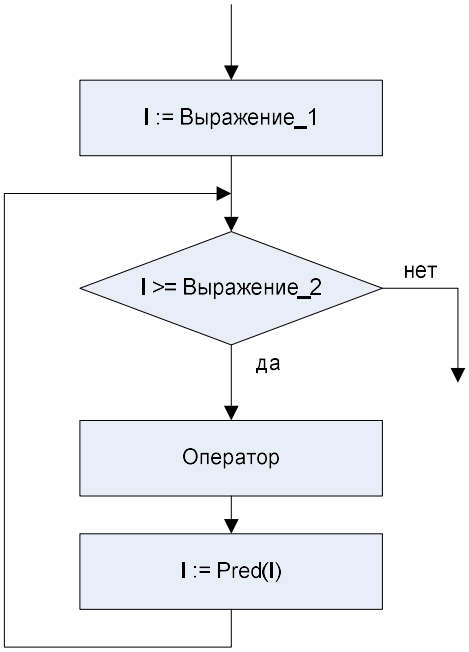
А что будет, если поменять границы диапазона местами так, чтобы перебор значений не имел смысла? Например, так:

```
for I := 10 to 1 do
  WriteLn(I:4, ' ^ 2 = ', (I * I):8);
```

Эксперимент показывает, а спецификация языка подтверждает: всё будет в порядке, просто тело цикла не выполнится ни разу. В общем виде алгоритм работы оператора for представляет собой цикл с предусловием:



Если диапазон задан через **to**



Если диапазон задан через **downto**

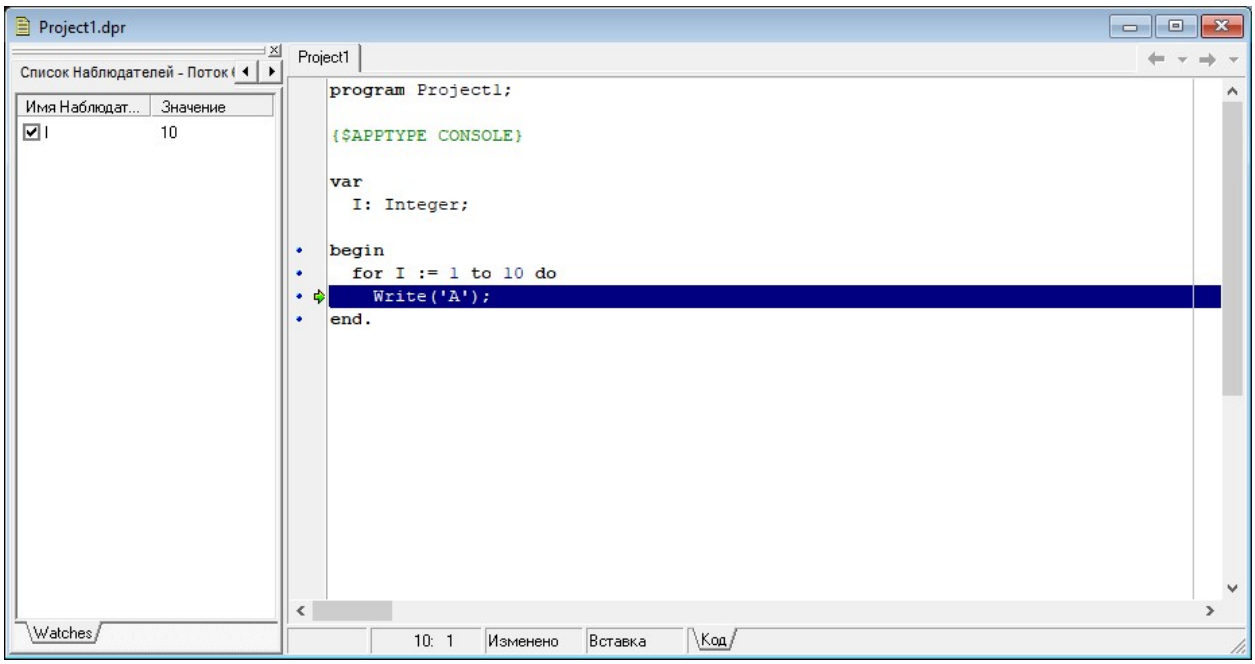
В приведённых схемах алгоритмов имеется неточность. Дело в том, что значения обоих выражений, задающих диапазон изменения параметра цикла, вычисляются один раз, до начала выполнения цикла. Таким образом, если значение `<Выражение_2>` будет изменяться в ходе выполнения цикла, это никак не повлияет на то, сколько раз он выполнится и какие значения будет принимать переменная цикла. Впрочем, это в любом случае не очень хорошая идея, которую Ваши коллеги бы явно не оценили.

Отличительной особенностью цикла for в Pascal и Delphi по сравнению с другими языками программирования является его ориентированность на максимальную оптимизацию решения типовой задачи. Дело в том, что во многих процессорах и контроллерах на аппаратном уровне поддерживаются специальные команды, позволяющие организовывать цикл со счётчиком максимально эффективно, однако эти инструкции, как правило, ведут подсчёт итераций с единичным шагом.

Иногда некоторые проявления этой особенности можно заметить. Попробуем выполнить пошагово такую программу:

```
program ForExample;

{$APPTYPE CONSOLE}

var
  I: Integer;

begin
  for I := 1 to 10 do
    Write('A');
end.
```

При просмотре значения переменной `I` окажется, что на первой итерации она оказывается равной 10, а не 0, как того требует текст программы:



Вместе с тем, это никак не влияет на правильность работы программы: вычисления в программе происходят с правильными значениями. Это явление вызвано совпадением двух факторов.

Во-первых, компилятор, опираясь на свойства for, заметил, что можно оптимизировать его выполнение: параметр цикла используется только для того, чтобы задать количество повторений, а само значение счётчика не участвует в вычислениях. В результате был сгенерирован машинный код, который использует более эффективный способ организации цикла — с обратным отсчётом. Более того, для переменной `I` даже не выделяется память!

Во-вторых, отладчик — модуль среды программирования, который отвечает за пошаговое выполнение, — в действительности работает не с исходным кодом, а с машинным, и ищет значения переменных там, где они должны находиться. Компилятор же применил логику изменения переменной-счётчика, которая отличается от прописанной в исходном коде, но честно указал, где хранится значение `I` во время работы программы. Отладчик честно отображает то значение, которое там находит.

Мораль этой басни такова: цикл for компилятором подвергается особенно агрессивным оптимизациям, поэтому внутренняя реализация может отличаться от ожидаемой. Из этого следует ещё два правила-ограничения в использовании оператора цикла for.

Значение переменной цикла for нельзя изменять в теле цикла

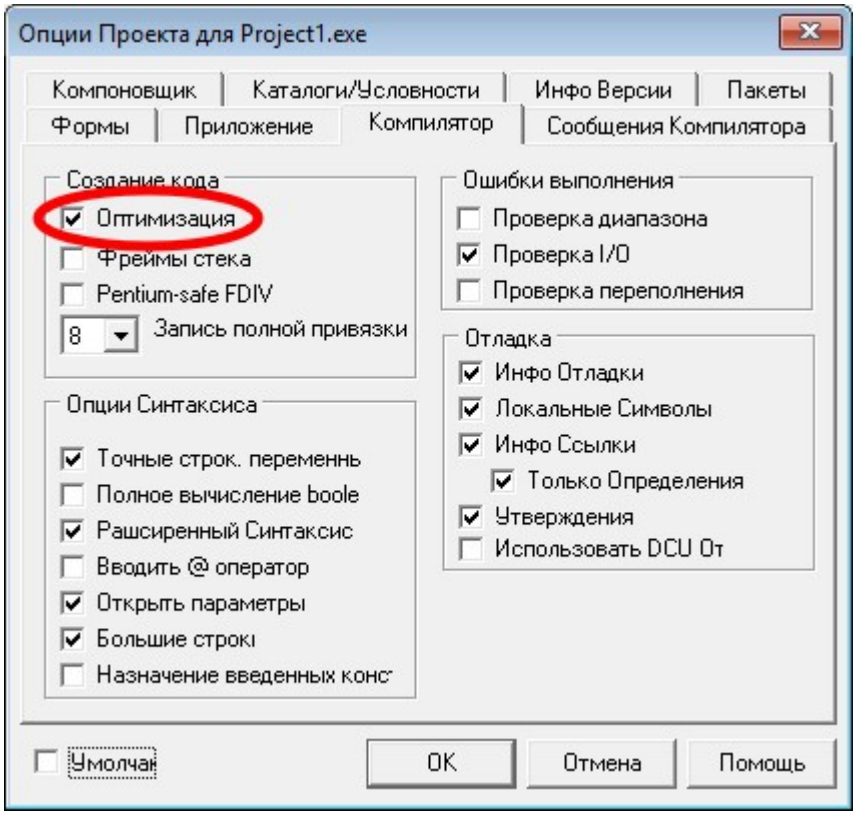
Причина проста: в лучшем случае это помешает компилятору применить оптимизацию, в худшем же оптимизация будет применена, но результат присваивания, задуманный программистом, не будет иметь смысла, т.к. реальный порядок изменения значения переменной цикла будет отличаться от предполагаемого программистом. Если в результате получится некорректно работающая программа или программа, которая работает корректно только при использовании определённого компилятора, версии компилятора или настроек компиляции — это будет считаться ошибкой программиста.

Логика здесь примерно такая же, как и с автомобилем: если Вы разогнались до 120 км/ч, а потом решили прямо на ходу пальцем выковырять камешки из протектора переднего колеса, то Вы можете быть миллион раз правы в своей любви к чистоте, но механизм работает на полную катушку. Либо остановите машину, либо не обижайтесь потом.

Значение переменной цикла for после окончания цикла не определено

Поскольку «под капотом» цикл for может быть перестроен компилятором так, как это покажется ему оптимальным, последнее значение, которое примет переменная цикла, может оказаться каким угодно и любые Ваши ожидания могут с этим значением не совпасть. Если даже здесь и сейчас Вам повезло и она оказалась равна, например, последнему значению из диапазона — в следующий раз и в других условиях всё может оказаться иначе.

Если Вы повадились тайком кормиться бутербродами из ящика стола Вашего коллеги по работе, не удивляйтесь, если однажды их там не окажется, и не обижайтесь, если однажды они окажутся с сюрпризом. В конце концов, Вы сами предложили ему занять этот стол.



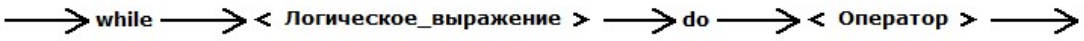
Но как быть, если Вам всё же нужно увидеть в отладчике те же значения, которые «видит» Ваша программа? Очень просто: отключите в настройках компилятора оптимизацию.

Операторы циклов на случай важных переговоров

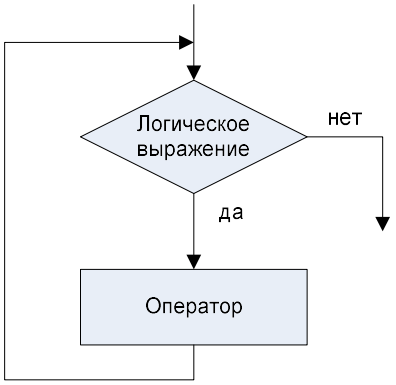
Предположим, что возможности оператора for слишком ограниченные для Вашей задачи. Ну, скажем, это итерационный цикл и вот так просто посчитать от 1 до 10 не получится. На этот случай имеется ещё два оператора цикла:

- оператор цикла с предусловием while;
- оператор цикла с постусловием repeat…until.

< Оператор_while > ::=



Логика работы *оператора цикла с предусловием* while соответствует следующей схеме алгоритма:



Попробуем написать программу, которая вычисляет значение выражения

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Перед нами типичная задача на использование итерационных циклов. Обратим внимание на то, что каждое последующее слагаемое получается из предыдущего домножением на x^2 и делением на очередные два натуральных числа: причём для второго слагаемого это числа 2 и 3, для третьего — 4 и 5 и т.д.

Зададимся некоторой точностью вычисления:

```
const
  Epsilon = 0.0001;
```

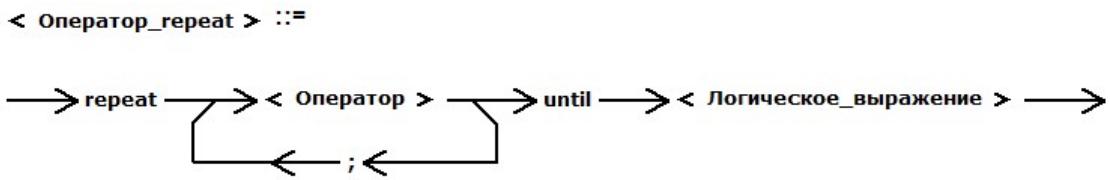
Для вычислений будет удобно, кроме переменной `X` для хранение аргумента функции (исходные данные) и `Y` для накопления результата, иметь переменную `Curr` для значения слагаемого на текущей итерации и вспомогательную переменную `N`, чтобы отслеживать, какое по счёту слагаемое будет вычисляться на текущей итерации:

```
var
  X, Y, Curr: Real;
  N: Integer;
```

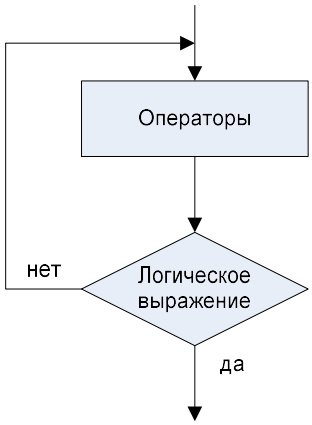
Сами вычисления в этом случае можно оформить следующим образом:

```
begin
  Write('Enter X: ');
  ReadLn(X);

  Curr := X;
  Y := X;
  N := 0;
  while Abs(Curr) >= Epsilon do
  begin
    Inc(N);
    Curr := Curr * X * X / (2 * N) / (2 * N + 1);
    Y := Y + Curr;
  end;

  WriteLn('Y = ', Y:0:6);
  ReadLn;
end.
```

Альтернативой оператору while является *оператор цикла с постусловием* repeat…until:



Принцип работы оператора repeat…until в точности соответствует классическому циклу с постусловием:



Отличительная особенность этого оператора заключается в том, что он позволяет записывать в теле цикла несколько операторов, разделённых точкой с запятой, без использования составного оператора.

Перепишем программу для вычисления по ранее приведённой формуле с использованием repeat…until:

```
begin
  Write('Enter X: ');
  ReadLn(X);

  Curr := X;
  Y := X;
  N := 0;
  repeat
    Inc(N);
    Curr := Curr * X * X / (2 * N) / (2 * N + 1);
    Y := Y + Curr;
  until Abs(Curr) < Epsilon;

  WriteLn('Y = ', Y:0:6);
  ReadLn;
end.
```

Как видим, отличия весьма несущественны.

Тонкости терминологии

Обратите внимание на то, что в терминологии, применяемой в отношении циклов и операторов для их организации, сложилась ситуация, в которой легко перепутать одно с другим. Следует чётко различать циклы — элемент алгоритма, т.е. идеи решения, — и операторы цикла — синтаксические конструкции в языках программирования.

В случае циклов имеется две независимые классификации. Как Вам уже должно быть известно, по местоположению условия цикла выделяют

- циклы с предусловием;
- циклы с постусловием;

а по виду условия

- циклы с параметром;
- итерационные циклы.

Что же касается операторов цикла, то в большинстве языков программирования имеется как минимум три оператора, позволяющих организовать цикл, и называют их обычно так:

- оператор цикла с предусловием (while);
- оператор цикла с постусловием (repeat…until);
- оператор цикла с параметром (for).

Очевидно, что первые два оператора могут использоваться как для организации циклов с параметром, так и для итерационных циклов, так что для операторов цикла мы имеем дело с одной классификацией, а не с двумя.

Дополнительные вопросы

1. В чём ошибка в следующем фрагменте программы?

```
for I := 5 to 20 do
begin
  Write(Chr(I + Ord('A')):2);
  I := I + 1;
end;
```

2. Почему нельзя точно сказать, что выведет данная программа?

```
I := 48;
for Digit := '0' to '9' do
begin
  I := I - Ord(Digit);
end;
WriteLn(Digit:2, I:6);
```

3. Что выведет следующая программа?

```
X := 0;
for I := 1 downto 10 do
  X := X + I * I / 2;
WriteLn(X:6);
```

4. Что выведет следующая программа?

```
X := 0;
I := 50;
while I <= 50 do
  X := X + I * I / 2;
WriteLn(X:6);
```

4. Что выведет следующая программа?

```
X := 0;
I := 50;
while I > 50 do
begin
  X := X + I * I / 2;
  I := I - 2;
end;
WriteLn(X:6);
```