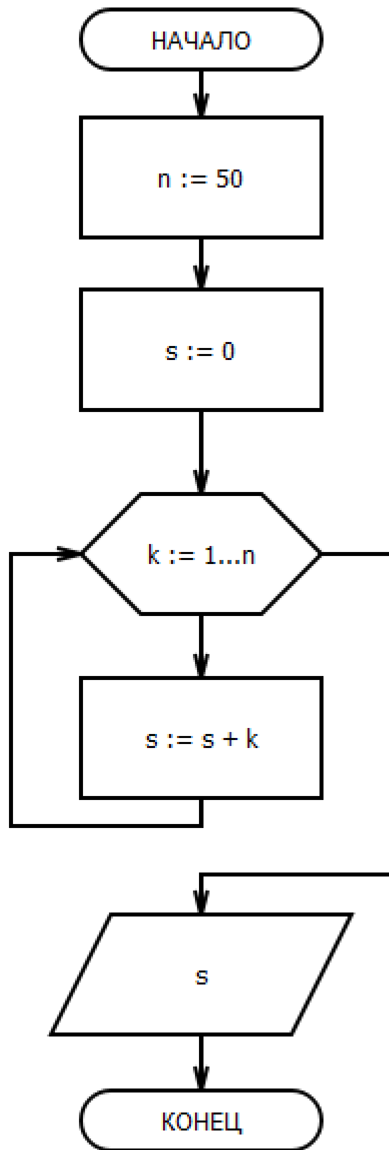
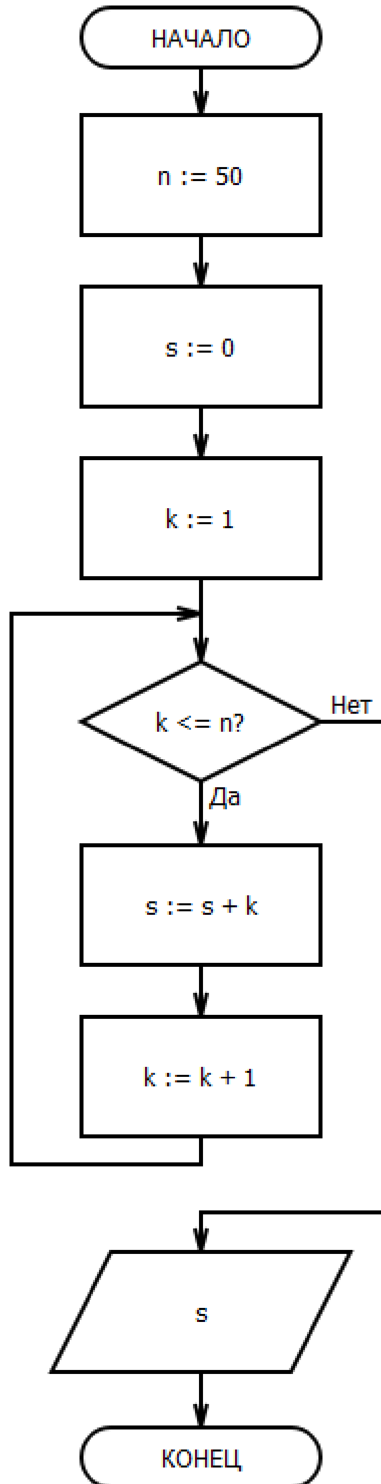
Циклы в Pascal

FOR



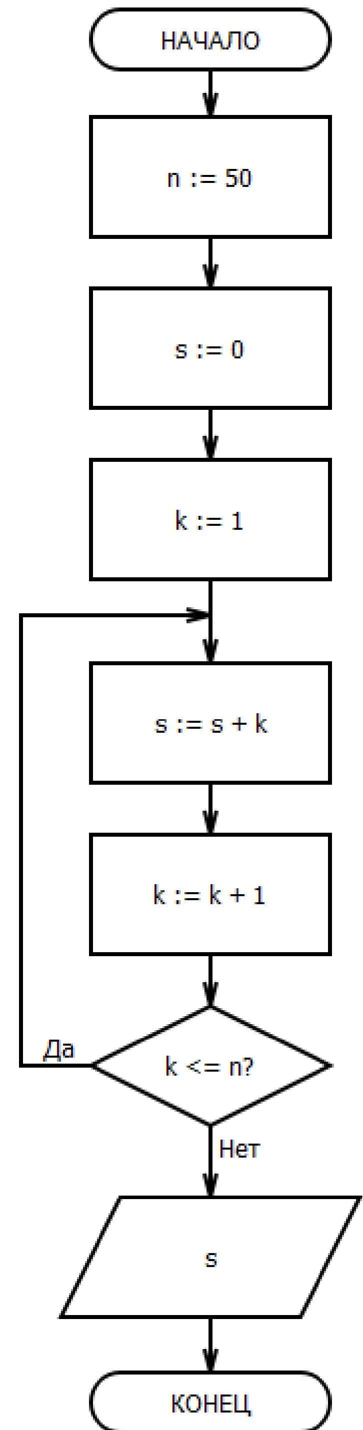

```
// находим сумму
var s, n : integer;
begin
  n := 50;
  s := 0;
  for var k := 1 to n do
  begin
    s := s + k;
  end;
  WriteLn(s);
end.
```

WHILE




```
// находим сумму
var s, n, k : integer;
begin
  n := 50;
  s := 0;
  k := 1;
  while k <= n do
  begin
    s := s + k;
    k := k + 1;
  end;
  WriteLn(s);
        end.
```

REPEAT



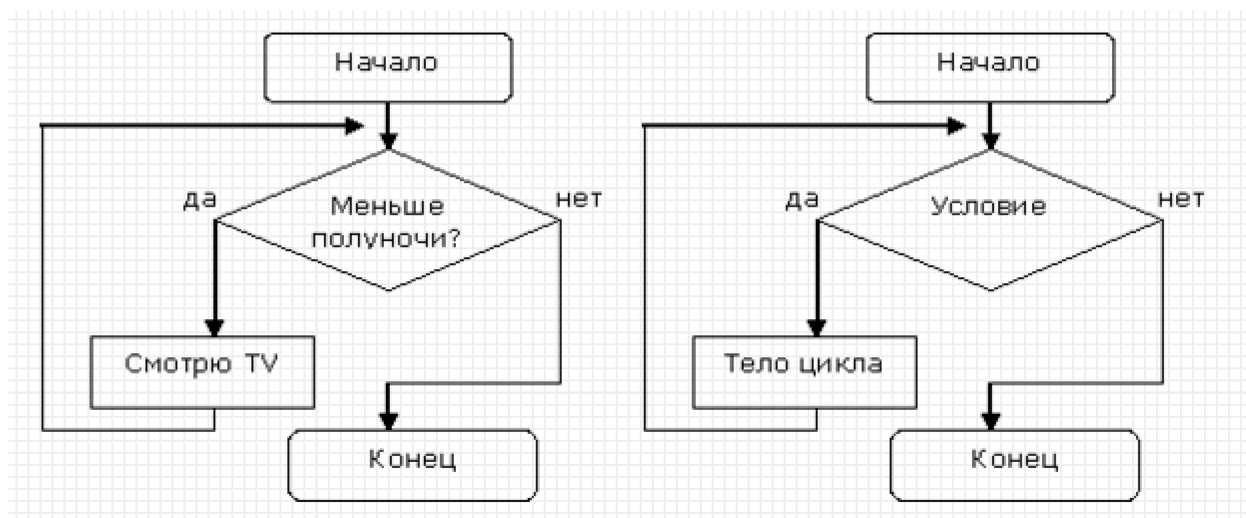

```
// находим сумму
var s, n, k : integer;
begin
  n := 50;
  s := 0;
  k := 1;
  repeat
    s := s + k;
    k := k + 1;
  until k > n;
  WriteLn(s);
end.
```

Оглавление

Операторы цикла For, While и Repeat 2
Цикл For 2
Цикл While 4
Цикл Repeat 6
Оператор присваивания 7
Отладка программ 8

Так как PascalABC.NET это не только язык программирования, но и редактор кода, и отладчик, и компилятор, то его называют Интегрированной Средой Разработки программ, или сокращенно **ИСР** *(по-английски - Integrated development environment, IDE).*

Вопрос: к какому виду цикла относится данный алгоритм?



Операторы цикла For, While и Repeat

Очень часто в программах встречаются ситуации, когда одни и те же действия нужно *повторить многократно.* Для таких случаев в языке паскаль припасены три оператора циклов - *For, While* и *Repeat-Until.*

Примеры циклов мы можем найти и в повседневной жизни:

- времена года (циклически сменяют друг друга весна - лето - осень - зима),
- время суток (утро - день - вечер - ночь),
- фазы Луны,
- вращение планет вокруг Солнца и Солнечной системы вокруг центра Галактики,
- спортивные состязания,
- учебный год,
- режим дня,
- дыхание и кровообращение,
- часы,
- биоритмы активности человека.

Цикл For

***Цикл For* используется в программах тогда, когда диапазон изменения какой-либо переменной точно известен.**

Цикл **For** для *единственного* оператора в теле записывается так:

```
For переменная_цикла:=начальное_значение to конечное _ значение do
Заголовок цикла
оператор1;         //Тело цикла
```

Цикл For для *блока операторов* записывается так:

```
    //Заголовок цикла
For переменная_цикла:=начальное_значение to конечное_значение do
begin
  оператор1;                          // Тело цикла
  оператор2;
  операторN;
end;                                 //Конец оператора цикла
```

А работает так:

1. Переменной цикла (она также называется *параметром* цикла) присваивается *начальное значение.*
2. Текущее значение переменной цикла сравнивается с её *конечным значением.* Если оно меньше конечного значения или равно ему, то последовательно выполняются операторы *оператор1, оператор2, ... , операторN,* составляющие *тело цикла.* Когда программа дойдёт до конца оператора цикла, она снова вернётся в заголовок, где переменная цикла получит следующее значение, на единицу больше текущего (то есть будет автоматически выполнен оператор присваивания **i := i + 1**). Значение переменной цикла увеличится на единицу, и цикл вернётся в начало *п.2.* Таким образом, переменная *i* последовательно принимает значения от **начального** до **конечного**, что нам и нужно. Если бы нам потребовалось повторить сто раз, то мы бы записали заголовок цикла так: ***For** i := 1 **to** 100 **do***

3. Если больше, то выполнение цикла заканчивается и управление переходит к следующей за ключевым словом ***end*** строке - для блока операторов или к строке, следующей за оператором в теле цикла - если он единственный.
4. **Количество повторов** = конечное_значение – начальное_значение + 1

При наборе цикла *for* удобно пользоваться шаблоном кода. Для единственного оператора в теле нажмите клавишу f, а затем *Shift+ПРОБЕЛ:*

```
for := to do
```

Для блока операторов - наберите буквы *fb* и нажмите клавиши *Shift+ПРОБЕЛ*:

```
for := to do
begin

end;
```

В качестве идентификатора *переменной цикла* часто выбирают короткие имена, часто из одной буквы - *i, j, k, l, m, n.*

Если начальное значение переменной цикла *больше* конечного, то цикл вообще не выполнится ни одного раза, а программа сразу перейдёт на строку, следующую за концом цикла.

Если какой-либо оператор в теле цикла использует переменную цикла, то её значение равно текущему. **Изменять значение переменной цикла в самом цикле нельзя!**

Иногда бывает необходимо, чтобы переменная цикла *уменьшала* своё значение. В этом случае вместо ключевого слова *to* в заголовке цикла следует записать *downto:*

```
For переменная_цикла:=начальное_значение downto конечное_значение do
```

Естественно в этом случае *начальное значение* должно быть не меньше *конечного,* иначе цикл не выполнится ни разу.

Если параметр цикла должен изменять своё значение на число, отличное от +/-1, то удобнее использовать циклы *while* или *repeat.*

В языке программирования *PascalABC.NET* допускается объявлять переменную цикла непосредственно в его заголовке. Это можно сделать двумя способами:

```
var переменная_цикла переменная := начальное значение        (1)
переменная_цикла переменная: тип := начальное значение
```

Такая переменная доступна только в теле цикла (локальная переменная цикла), а после его окончания она уничтожается. Мы будем применять способ (1).

```
for var i := 5 to 20 do
```

```
// вопрос: что вычисляется в цикле?
var p, n : integer;
begin
  n := 5;
  p := 1;
  for var k := 1 to n do
  begin
    p := p * k;
  end;
  WriteLn(p);
end.
```

Цикл While

While [пока) называется оператором цикла с *предусловием,* а *Repeat (повторять)* - оператором цикла с *постусловием.* Он отличается от цикла *While* только тем, что проверка завершения цикла проводится в конце цикла, а не в его начале. Это значит, что цикл *Repeat* всегда выполняется хотя бы один раз, а цикл *While* может вообще не выполняться ни разу. Различаются эти циклы очень мало.

Цикл While с *единственным* оператором в теле цикла *записывается* так:

```
While условие_выполнения_цикла do // Заголовок цикла
 оператор1;  // Тело цикла
```

А с *блоком* операторов - так:

```
While условие_выполнения_цикла do   // Заголовок цикла
 begin
  оператор1;      // Тело цикла
  оператор2;
  операторN;
end;     ///Конец оператора цикла
```

А работает *он так:*

1. *Проверяется* условие выполнения цикла**.**
2. Если оно *ложно* (не выполняется), то цикл заканчивается, и программа переходит на строчку, следующую за ключевым словом *end***.**
3. Если оно *истинно***,** то последовательно выполняются операторы *оператор1, оператор2***,** *...* , *операторN***,** которые называются *телом цикла,* и управление передаётся в *п.1***.**

Поскольку условие проверяется *до* тела цикла, то операторы могут вообще не выполняться - если выражение с самого начала ложно.

При наборе цикла *while* удобно пользоваться шаблоном кода. Для единственного оператора в теле нажмите клавишу w, а затем *Shift+ПРОБЕЛ:*

```
while do
```

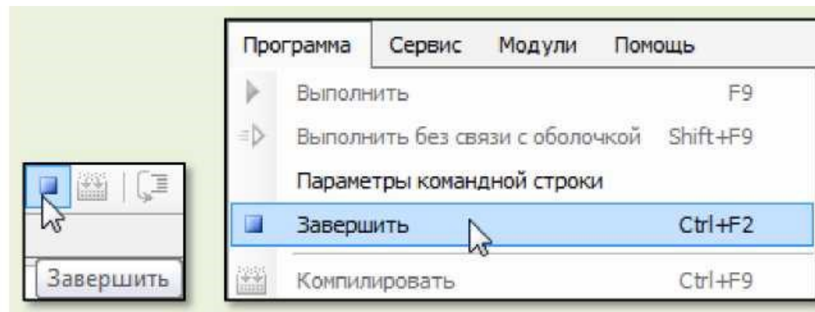
Для блока операторов - наберите буквы *wb* и нажмите клавиши *Shift+ПРОБЕЛ:*

```
while  do
 begin

 end;
```

Как мы видим, цикл заканчивается тогда, когда условие станет ложным. А почему *изменяется* значение *условного выражения* в заголовке цикла? - Да потому, что изменились значения переменных, входящих в это условие. **И это может произойти только в теле цикла**. Отсюда вывод: цикл *While* может вообще никогда не закончиться, и программа зациклится навечно, что очень часто и бывает! Не томите компьютер и закройте программу.

При работе со стандартной консолью в режиме отладки для этого следует нажать кнопку *Завершить* (Рис., слева), или выполнить команду меню *Программа* > *Завершить* (Рис., справа), или нажать клавиши *Ctrl+F2.*



Закрываем зациклившуюся программу

Всегда следите за изменением *переменной* (или переменных), входящей в условие выполнения цикла! Если она не изменяется, то цикл либо не выполнится ни одного раза, либо будет выполняться бесконечно! Обратите внимание на отличие цикла *While* от цикла *For,* в котором, наоборот, переменную цикла изменять нельзя.

Применяйте цикл *For,* если число повторений известно заранее, а в противном случае лучше подойдёт цикл *While или Repeat.*

Паскаль позволяет записывать в строке больше одного оператора, поэтому вы можете вместо двух строчек:

```
n := 0;
Number := '';
```

«вбить» код в одну:

```
n := 0; Number := '';
```

Обычно этого делать не следует: тогда исходный код будет гораздо легче понять.

Цикл Repeat

Как мы уже знаем, *Repeat* - называют оператором цикла с *постусловием.* В нём проверка завершения цикла проводится в *конце* цикла, а не в его начале. Это значит, что цикл *Repeat* всегда выполняется хотя бы один раз.

Цикл Repeat записывается так:

```
Repeat                                    //Начало цикла
  оператор1;                               //Тело цикла
  оператор2;
  …
  операторN;
Until условие_завершения_цикла;    // Конец оператора цикла
```

А работает *так:*

1. Последовательно выполняются операторы *оператор1, оператор2, …, операторN* в *теле* цикла.
2. *Проверяется* условие выполнения цикла после ключевого слова until.
3. Если оно *истинно* (выполняется), то цикл заканчивается, и программа переходит на строчку, следующую за концом оператора цикла.
4. Если *условие* ложно (не выполняется), то управление передаётся в *п.1.*

Итак, давайте сравним операторы *while* и *repeat.*

1. В цикле *while* условие выполнения цикла проверяется до тела цикла, а в цикле *repeat* - после. Поэтому первый цикл может вообще не выполняться, а второй выполняется, по крайней мере, один раз.
2. Ключевые слова *repeat - until* играют роль операторных скобок, поэтому в теле цикла может быть любое число операторов, которые не нужно заключать в операторные скобки *begin - end.*
3. Цикл *while* выполняется, пока условие *истинно,* а цикл *repeat* - пока оно *ложно.*
4. Любой цикл *repeat* можно заменить циклом *while,* но не наоборот!
5. Цикл *repeat* также может стать *бесконечным,* если условие всегда остаётся ложным.

При наборе цикла *repeat* удобно пользоваться шаблоном кода. Нажми клавишу **r**, а затем *Shift+ПРОБЕЛ:*

```
Repeat

Until  ;
```

Оператор присваивания

Самый «востребованный» оператор в любом языке программирования - это, безусловно, **оператор присваивания**.

Он записывается так:

```
var переменная := выражение;
```

Переменная - это простая переменная, элемент массива или свойство объекта. В результате выполнения этого оператора значение переменной станет равным значению выражения. Выражением может быть другая переменная, константа, константное выражение или функция. Сложные выражения могут быть составлены из переменных и констант, соединённых знаками арифметических операций.

Например:

Оператор присваивания	**Результат присваивания, значение переменной**
`var Width := 100;`	100
`var Height := Width + 20;`	120
`var Mas: array[1..20] of string;`	
`Mas[12] := 'Двенадцать';`	Двенадцать
`Mas[13] := 'Двенадцать' + 1.ToString();`	Двенадцать1
`var Str := Mas[13] + Height.ToString();`	Двенадцать1120
`var maxNum := Max(1,11);)`	11

Различие между оператором присваивания **:=** и знаком равенства **=** в математике можно показать на примере. Очень часто в программах можно встретить вот такие выражения:

```
var x := 10;
x := x + 1;
```

Первая строка не вызывает никаких возражений: согласно правилам алгебры, неизвестная величина *x* равняется 10. Во второй строке *сначала* вычисляется значение выражения *справа* от знака присваивания, а *затем* оно *присваивается* переменной *слева* от знака присваивания. В нашем случае: в первой строке переменная *x* получила значение 10. Вычисляем значение выражения во второй строке: 10 + 1 = 11. После выполнения оператора присваивания переменная *x* будет равна 11. Конечно, выражение

```
x = x + 1;
```

в математике было бы неверным. Поэтому в паскале

- знак равенства = применяется в условных выражениях при сравнении двух значений,
- а для присваивания специально введён знак :=, чтобы не путать его со знаком равенства.

Отладка программ

Лови жучков!
Программистская пословица

Отладка программ - самый трудный и ответственный момент в программировании. В самом деле, если хотя бы одна ошибка останется в готовом приложении, то оно иногда будет работать неправильно!

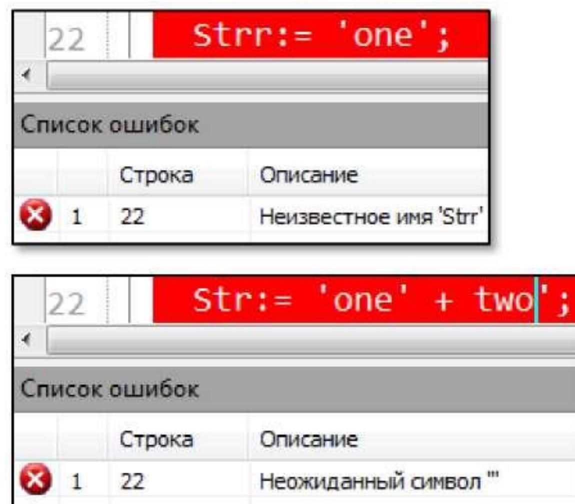
> *Курьёзный случай*. Клиент одного банка ежемесячно получал требование: погасить долг в течение месяца. Однако сумма долга составляла 0 долларов, 0 центов. Конечно, оплачивать такой счет не имело никакого смысла. Этот пресловутый клиент именно так и думал. И каждый месяц получал новое предупреждение. Так продолжалось до тех пор, пока он не оплатил «долг», выслав банку чек на означенную сумму. А дело тут в том, что компьютерная программа в банке не учитывала, что нулевой долг таковым не является.

Ошибки, которые проявляются только при работе уже готовой программы, принято называть словом *баг* - от английского *bug* - *жук.*

Этот термин возник в 1945 году, когда небольшая бабочка, незаконно проникшая в вычислительную машину, замкнула реле, чем вызвала ошибки в её работе. С тех пор слово *debugging,* то есть *извлечение скрытых багов,* означает отладку программы.

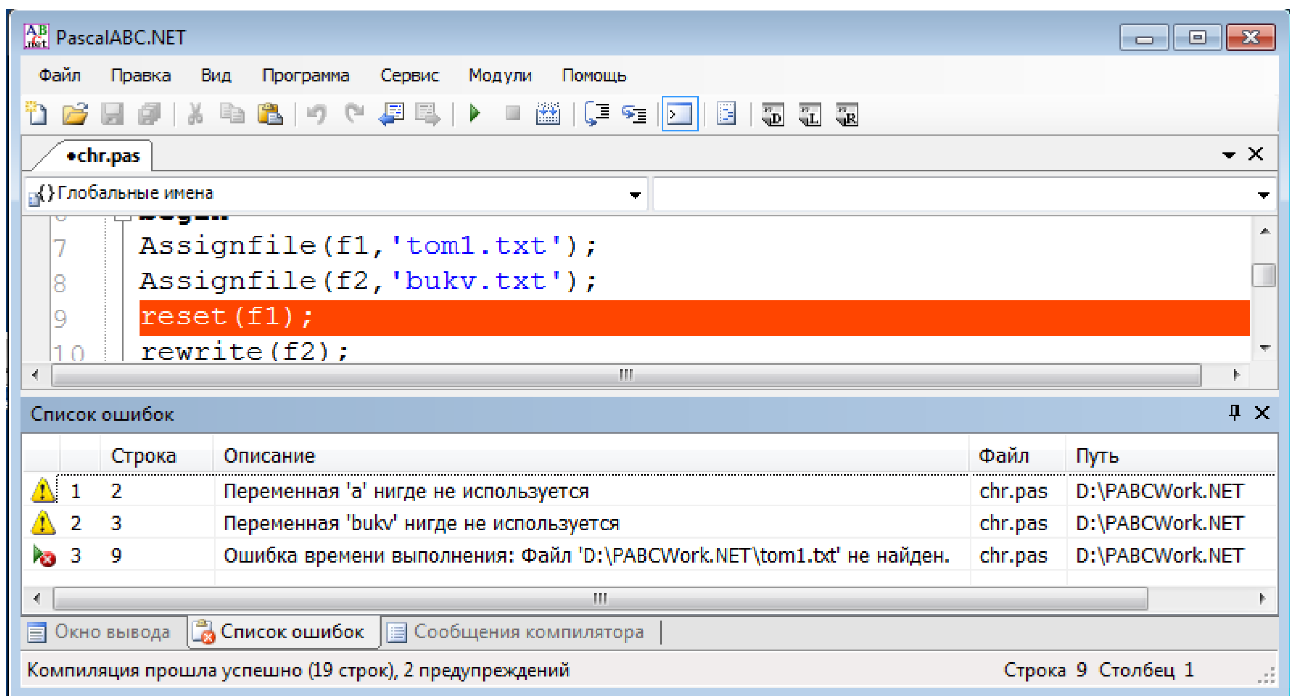
Все ошибки в программе можно условно разделить на синтаксические и логические.

Синтаксические (грамматические) ошибки появляются обычно из-за невнимательности или описки. Такие ошибки нетрудно «обезвредить», поскольку *ИСР* в процессе компиляции легко находит их и сообщает об этом в окне *Список ошибок,* которое находится под окном *Редактора кода*.



Отладчик нашёл ошибку

Кроме ошибок, отладчик выдаёт и *предупреждения* (Рис). Их можно и проигнорировать, поскольку такие действия программы не нарушают её работоспособности, но желательно принимать их во внимание.



Отладчик нашёл неиспользуемые переменные
и не нашел файл tom1.txt

Достаточно щёлкнуть по сообщению, и курсор установится в то место исходного кода, где найдена ошибка. Строка с ошибкой также выделяется **красным ц**ветом.

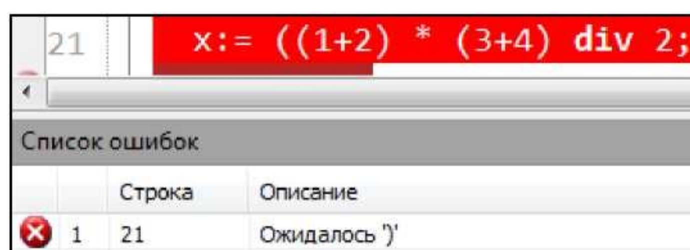
Номер строки с ошибкой указываются слева от описания ошибки.

Некоторые ошибки легко распознать благодаря *окрашиванию* элементов программы в разные цвета и по атрибутам шрифта. Например, ключевые слова выделяются **жирным** шрифтом (Рис., слева). А если их написать неверно, то шрифт останется обычным (Рис., справа).

Выделение ключевых слов

Таким образом, при должной внимательности можно легко найти синтаксические ошибки.

Обычные синтаксические ошибки - это неверно написанные идентификаторы, непарные скобки и кавычки, пропуск точки с запятой в конце оператора и двоеточия в знаке присваивания.



Типичные синтаксические ошибки

Логические ошибки возникают вследствие неправильного алгоритма или неверной записи операторов программы, которые компилятор *паскаля* найти не может, потому что они не противоречат правилам языка. Логические ошибки отыскать очень трудно, а сам поиск таких ошибок называется *отладкой программы*.

В первую очередь, при написании программы следует помнить, что после добавления логически законченного фрагмента кода, сразу же следует проверить, правильно ли он работает. Никогда не пишите большие куски кода, иначе найти ошибки будет очень трудно.

Отлаженный промежуточный вариант программы всегда сохраняйте на диске, добавляя к названию программ номер версии: *MyProgram1, MyProgram2* и так далее. Конечно, последние изменения в тексте легко удалить с помощью кнопки *Отмена*, но сможете ли вы всегда вернуться к тому коду, который предшествовал ошибке? - Поэтому не жалейте времени на сохранение отлаженного кода!

Отладчик *ИСР* действует только при запуске программы в *отладочном* режиме с помощью кнопки *F9*. Из этого следует, что консольные приложения с модулем *CRT* необходимо отлаживать самостоятельно. Для этого в критических местах программы нужно вставлять операторы, которые выводят отладочную информацию в окно программы, т.е. команду **writeln**.

Аналогично вы можете контролировать значения любых переменных программы, вставляя отладочные операторы в те места кода, где переменные должны изменять свои значения.

После отлова всех жучков отладочные операторы можно удалить или закомментировать. Второй вариант предпочтительнее, так как исходный код может ещё вам пригодиться при написания другой программы, а в ней также могут понадобиться отладочные операторы. Их можно и снова написать, но раскомментировать строку гораздо быстрее и удобнее.

```
Файл  Правка  Вид  Программа  Сервис  Модули  Помощь
chr.pas   •задача А 2013-2014 repeat.pas
Глобальные имена
42  //    for i := 0 to n do
43        stepenk := stepenk * k;
44      case stepenk of
          1..31: begin mes := 1;den := stepenk - 0; end; //янв
          32..59: begin mes := 2;den := stepenk - 31; end;
47        60..90: begin mes := 3;den := stepenk - 59; end;
48        91..120: begin mes := 4;den := stepenk - 90; end;
49        121..151: begin mes := 5;den := stepenk - 120; end;
50        152..181: begin mes := 6;den := stepenk - 151; end; /
51        182..212: begin mes := 7;den := stepenk - 181; end;
52        213..243: begin mes := 7;den := stepenk - 212; end;//
Компиляция прошла успешно (35 строк), 1 предупреждений      Строка 52  Столбец 6
```

Точка останова программы

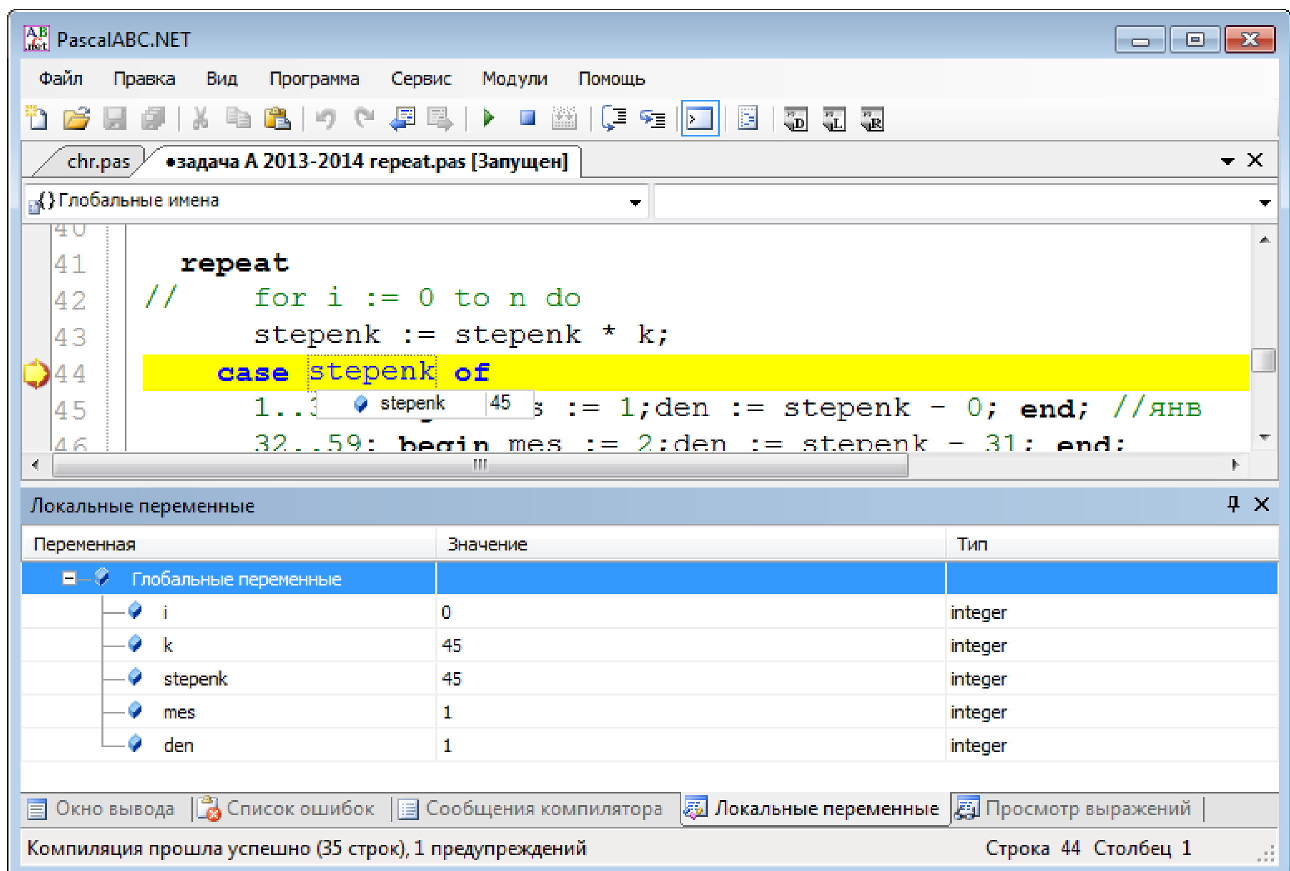
Пусть мы хотим понаблюдать за изменением значения переменной stepenk. Находим

строку и ставим, кликнув мышкой, красную точку перед номером строки. Так мы задаём *точку останова* программы при её выполнении.

Как только программа перейдёт на эту строку, её выполнение приостановится. Строка будет выделена жёлтым цветом и стрелкой (Рис.). Подведя курсор мышки к переменной, мы сможем прочитать в подсказке её текущее значение (в данном примере - *при* выполнении оператора в этой строке)

Точно так же вы можете исследовать и другие переменные программы.

Значение переменной в подсказке



Все переменные программы в одном окне

Ещё больше информации предоставляет нам окно *Локальные переменные,* которое появляется при остановке программы.

В нём вы легко найдёте текущие значения всех переменных программы - как локальных, так и глобальных.

Чтобы *убрать* точку останова, нужно просто кликнуть по ней мышкой.

После приостановки программы нажмите кнопку *Выполнить* (F9), чтобы продолжить её работу.