ИНФОРМАТИКА

Комлев Н. Ю.

«СОЛОН»

Самоучитель игры на ПАСКАЛЬ

ABC и немного Турбо


```
TArray3x3 = array[1..3,1..3] of integer;
var
  a                : TArray3x3;
  x,y              : integer;
  driver, mode     : integer;
  sc               : byte;
  result           : integer;
  gamover          : boolean;
  xkuda, ykuda     : integer;
  i,j,k            : integer;
begin

  driver:=VGA;
  mode:=VGAHi;
  InitGraph( driver, mode, '');

  DrawField;
  x:=1;
  y:=1;
  DrawCursor(x,y);
  for i:=1 to 3 do
    for j:=1 to 3 do
      a[i,j]:=0;
  gamover:=false;

  repeat
    if OurKeyPressed then begin
      sc:=OurReadKey;
      if (sc = ArrowLeft) and (x>=2) then begin
        HideCursor(x,y);
        x:=x - 1;
        DrawCursor(x,y);
      end;
      if (sc = ArrowRight) and (x<=2) then begin
        HideCursor(x,y);
        x:=x + 1;
        DrawCursor(x,y);
```

ISBN 5-91359-112-8
9 785913 591128

ЭЛЕКТИВНЫЙ КУРС ✸ ПРОФИЛЬНОЕ ОБУЧЕНИЕ

УДК 681.3
ББК 32.97
К 63

Комлев Н. Ю.
Самоучитель игры на Паскале. ABC и немного Турбо. — М.: СОЛОН-ПРЕСС, 2013. —256 с.: ил.

С абсолютного нуля до полного овладения языком. Популярный в школах Pascal ABC и вечно живой Turbo Pascal. Весь Паскаль, среда программирования, технология разработки и отладки. Примеры программ — простых и не очень. Для младших студентов, старших школьников и всех, кто хочет программировать.

Виртуальный диск ***в помощь читателям книги с текстами всех программ длиннее десяти строк в двух версиях – для Pascal ABC и Turbo Pascal.***
Специально для Turbo Pascal (TP): установка, настройка, русификация, процесс отладки в Turbo Pascal, особенности работы с графикой, работа с клавиатурой в TP на уровне скан-кодов, полезные советы

ISBN 978-5-91359-112-8

Глава 2.
Переменные

Что такое и зачем

А теперь о главном. Три вещи, по нарастающей сложности, которые надо понять зарождающемуся программисту - переменные, массивы и указатели. Всё остальное - сущая ерунда. Сейчас будут переменные.
Пишем вот такую программу:

```
program kuku;
  uses
      Crt;
  var
      x                 : integer;
begin
end.
```

Что мы сделали? Объявили переменную. Имя у неё “x”. Тип у неё целый.
А теперь делаем вот так:

```
x:=5;
Writeln(x);
```

Транслируем, запускаем, смотрим, что получилось. На экране получилось вот что –
5
А теперь медленно: мы

- объявили переменную
- присвоили ей значение
- вывели значение переменной.

А что такое переменная, собственно? Вопрос из категории – а что такое электричество?
Переменная – ключевое понятие программирования. Без неё не обходится ни один язык. Ну, почти ни один. (Здесь, и далее, а также и прежде под языками я понимаю только и исключительно языки программирования).

Переменная имеет две составляющих – имя и значение. Как известно, на заборе что-то там написано, а лежат там дрова. Так вот, то, что написано – это имя переменной, а дрова – это значение.
В большинстве языков переменная имеет ещё и третью составляющую – тип.
В нашем случае тип переменной – целое число. То есть поместить в неё можно только целое число, дробное – никак, не говоря уже о всякой экзотике. Но к экзотике ещё вернёмся, попозже.

Для любознательных - тип переменной integer позволяет хранить значения от +2147483647 до -2147483648. Причина - его размер четыре байта. Вам мало? Даже не знаю, что и посоветовать.

x : integer; - это объявление переменной. Слева от двоеточия имя переменной. Часто говорят об идентификаторе переменной. Это то же самое, что имя. Имя может быть написано большими буквами или маленькими – Паскалю всё равно.
Два подряд идущих символа “:=” называются оператором присваивания. Слева от него имя переменной, справа – значение, которой ей присваивается.
А теперь сделаем с переменными что-нибудь полезное. Для начала объявим три переменные:

```
var
    x,y,z       : integer;
```

Двум из них присвоим значения:

```
    x:=5;
    y:=2;
```

А почему только двум? Потому что значение третьей переменной мы сосчитаем. На самом деле не мы, конечно, а компьютер.

```
x:=5;
y:=2;
z:=x+y;
Writeln(z);
```

Получили вполне ожидаемый результат – на экране красуется семёрка.

А теперь быстро осваиваем вычитание и умножение.

```
z:=x-y;
Writeln(z);
```

```
z:=x*y;
Writeln(z);
```

Соответственно, в результате имеем два и десять.
Осталось четвертое действие – деление. С ним сложнее. Почему? Потому что числа у нас целые, а результат деление целым быть не обязан – от деления пятерки на двойку получим два с половиной. Что делать? Вспомнить чудное детство в первом классе – деление нацело с остатком. Делим пять на два – получаем два и один в остатке. Попрактикуйтесь в этом деле – как ни странно, некоторые напрочь это забыли.
Так что вместо одной операции получаем две – деление нацело и получение остатка.

Сначала деление нацело.

```
x:=5;
y:=2;
z:=x div y;
Writeln(z);
```

В результате имеем два.
А теперь получение остатка

```
x:=5;
y:=2;
z:=x mod y;
Writeln(z);
```

В ответе получаем единицу.
Задача посложнее – разобрать двузначное число на составляющие его цифры.

```
var
   x,c1,c2        : integer;
begin
      x:=85;
      c1:=x div 10;
      c2:=x mod 10;
      Writeln(c1);
      Writeln(c2);
end.
```

Маленькое новшество – имена переменных у нас уже не из одного символа, а из двух, причём второй символ – цифра. Главное, чтобы имя начиналось с буквы.
Обратите внимание на особый смысл выражения x **mod** 2. Если оно равно нулю, то число x четное, если единице – то нечетное. Это пригодится далее.
Переменные можно было бы объявить и каждую в отдельной строке:

```
var
  x              : integer;
  c1             : integer;
  x2             : integer;
```

Но мы написали как короче. Аналогично можно сэкономить и на операторах Writeln, объединив два в один.
Пишем

```
      Writeln(c1,c2);
```

Запускаем программу на выполнение и получаем не совсем то, что бы хотелось. Циферки слиплись. С одной стороны, всё правильно – вот восьмерка, вот пятерка – но хотелось бы как-то видеть, где кончается одно число, и где начинается другое. Напишем вот так:

```
      Writeln(c1:3,c2:3);
```

Гораздо лучше. Тройка после двоеточия означает, что мы требуем выделения для вывода числа трех символов. Число у нас всего из одной цифры, так что спереди перед ней появились два пробела.

А почему мы должны, глядя на результат работы программы, догадываться, где какой переменной значение? Делаем так:

```
Writeln('c1=', c1:3, 'c2=', c2:3);
```

То, что в кавычках, будет выведено "как есть", это мы уже проходили.
Долго думаем о тонкой разнице между c1 в кавычках и c1 без кавычек. В кавычках это просто бессмысленный набор символов. Компьютер выведет его так, как он написан. А без кавычек это – имя переменной, и выведено будет её значение. Если это сразу кажется понятным и очевидным – поздравляю…
Вы, конечно, уже заметили, что вывод всё равно не совсем такой, как хотелось бы – опять чего-то слиплось. Но теперь уж постарайтесь справиться сами.

Ввод и вывод

Напишем программу возведения числа в квадрат.

```
program Square;
  uses
      Crt;
  var
      x, x2        : integer;
begin
      x:=5;
      x2:=x*x;
      Writeln('x2=',x2);
end.
```

Всё хорошо, всё работает – пятью пять двадцать пять. А если надо шестью шесть тридцать шесть? Менять текст программы, снова компилировать - кажется как-то не очень правильным. Хотелось бы, что бы программа при

запуске спросила, а какое, собственно, число нам надо возвести в квадрат. Легко.
Вместо

```
x:=5;
```

пишем

```
Readln(x).
```

Теперь при запуске программы она чего-то ждёт. Программа упорно ждёт, когда мы наберем на клавиатуре число, хотя бы ту же пятерку, и нажмем **Enter**. Тоже как-то не очень здорово. Мы-то, конечно, знаем, что надо делать, а если вдруг кто непонятливый к компьютеру подойдёт?
Опять улучшаем программу. Пишем:

```
Write('x = ');
Readln(x).
```

Обратите внимание, что написано не Writeln, а именно Write. Исключительно из эстетических побуждений. Иначе мы бы перешли на новую строку и уже там набрали бы вводимое число. Число ввелось бы, конечно, но выглядит это как-то не очень аккуратно. Ещё обратите внимание на пробел после равенства. Исключительно из тех же эстетических соображений.
В результате наконец-то получили первую хоть сколько-то полезную программу.
Делаем вывод (извините, если для кого-то банальный) - порядочная программа должна:

- ввести данные (предварительно предложив их ввести)
- что-то с ними сделать
- вывести результат.

А теперь немного о грустном. Мы пока что работаем только с целыми числами. Если попытаться ввести вместо пяти пять с половиной, результат будет катастрофическим (попробуйте). Если в процессе ввода

промахнуться мимо нужной клавиши – та же самая катастрофа. Что делать? С непопаданием по клавишам – пока ничего. Внимательнее смотрите на клавиатуру. Разберёмся с этой проблемой попозже.

Дроби

Так что насчёт пяти с половиной? Это не просто, а очень просто. Меняем всего одну строку.
Вместо

```
var
  x         : integer;
```

пишем

```
var
  x         : real;
```

и наступает полное дробное счастье – ну почти.
Почему счастье не совсем полное?

Real - дробное число, занимающее восемь байтов. Дробные числа обычно называют плавающими или числами с плавающей точкой. Когда-то, давным-давно, были ещё и числа с фиксированной точкой, например, три знака до точки, два после. Числа с плавающей точкой отличались от них тем, что помнили определенное количество знаков(real - пятнадцать, шестнадцать, как повезёт) - а точка могла стоять где угодно. То есть, плавающее число помнит, к примеру, шесть знаков - 123456, но это может быть 123.456 или 0.0123456 или 123456000. А седьмой знак тут уже никак поместится не может.
Обратите внимание, что у целых переменных при изменении количества байт на число меняется максимальная величина, которое это число может содержать, а у дробных – ещё и точность представления числа.

Во-первых, дробные числа вводятся (и выводятся) с точкой в качестве разделителя. То есть, пресловутые пять с половиной будут выглядеть так: 5.5.

В качестве разделителя целой и дробной части в Паскале используется всегда точка, независимо от того, какой разделитель установлен в Windows.

Во-вторых, оператор Writeln(x) выведет на экран что-то страшное и ужасное. На самом деле, никакое оно не страшное и, даже, где-то полезное, но разбираться с ним (пока, по крайней мере) не будем. Будем улучшать.
Напишем

```
Writeln(x:8:2);
```

Очень напоминает вывод целых чисел – там тоже первая (и единственная) цифра задавала общее количество выводимых цифр. Если реально цифр было меньше, спереди выводимое число дополнялось пробелами. Но сейчас для нас интереснее вторая цифра. Двойка указывает, сколько знаков после десятичной точки должно быть выведено.
И опять очень сложный момент. Восьмерка здесь – сколько знаков всего выводится – до точки плюс после точки плюс один знак на саму точку. Почему-то очень многие ожидают, что восемь – это именно и только число знаков до точки.

А теперь у нас появилась ещё одна арифметическая операция – обычное деление, не нацело. Обозначается оно так:

```
x:=y/z;
```

Ещё одна программа, с небольшим отличием. В ней используется константа, то есть число, значение которого мы заранее знаем и, в процессе выполнения программы, измениться это значение не может. Чем она (константа), собственно, и отличается от переменной.

Если мне не изменяет память, константы делятся на константы именованные и неименованные. То есть, термины могут быть другими, но смысл остаётся. Сейчас мы встретимся с неименованной константой, а до именованных доберёмся в процессе освоения массивов. Точнее, формально константы в программах у нас уже были и раньше, но применять это гордое имя к простому числу «5» как-то недостойно.

```
program Circle;
  uses
      Crt;
  var
      R, S          : real;
begin
      Writeln( ‘Radius = ‘);
```

```
        Readln(R);
        S:=3.14*R*R;
        Writeln('S=',S:8:2);
end.
```

Здесь 3.14 – число пи – константа.

Идею поняли. А теперь – быстро пишем программу, которая считает, что получится, если положить в банк X рублей под Y процентов на один год. Для тех, кто совсем никак, дальше искомая программа. Но всё-таки, постарайтесь уж сами.

```
program Money;
  uses
        Crt;
  var
        Rub, rate, HowMany  : real;
begin
        Writeln( 'Rubles = ');
        Readln(Rub);
        Writeln( 'Rate = ');
        Readln(rate);

        HowMany:=Rub + (Rub*rate)/100;
        Writeln('Result=', HowMany:8:2);
        Readln;
end.
```

И ещё немного математики. В школе кроме четырех действий что-то говорилось ещё и про пятое – логарифмирование. Вспоминается также и тригонометрия с синусами, косинусами и прочими тригонометрическими функциями. В Паскале это всё есть и выглядит очень похоже на обычные математические формулы:

```
        y:=sin(x);
        y:=cos(x);
        y:=ln(x);
```

И называется это так же, как и в математике – функции. В Паскале есть много уже готовых функций, а попозже будем осваивать самостоятельное их изготовление.

И ещё о важном. Например, у нас есть объявление переменных

```
var
    x       : real;
    n       : integer;
```

В начале нашей программы следуют вот такие операторы присваивания:

```
x:=3.5;
n:=4;
```

До сих пор всё хорошо и понятно.

Теперь, если мы напишем

```
x:=n;
```

по-прежнему всё будет хорошо.
Но если мы напишем

```
n:=x;
```

то мы даже не дойдем до этапа выполнения программы. В процессе трансляции нам сообщат, что мы глубоко не правы:

«Ошибка: Попытка присвоить переменной типа integer значение переменной типа real».

Это надо постараться или запомнить, или как-то понять.
Чтобы запомнить – учим мантру: дробному целое присвоить можно, целому дробное присвоить нельзя.

Постараться понять – на уровне для блондинок: ну, у дробного есть ящички под целые и дробные части, а у целого – только один ящичек – для части целой. Поэтому, если мы пихаем целое в дробное, то оно из одного ящичка в два влезет, а если, наоборот, из двух ящичков в один – ну никак.

А вообще всё даже проще – в Паскале в принципе нельзя присваивать друг другу переменные разных типов. Таков Великий Замысел Отцов-Основателей. Единственная данная ими поблажка – дробному числу можно присвоить целое. Вот и всё.

TP. В чём разница

Да почти ни в чём. Точнее, две разницы, одна большая, другая поменьше.
Большая разница в том, что переменная типа integer вмещает в TP всего-навсего целое число от +32767 до -32768. А если вы хотите получить такую же вместимость, как в Pascal ABC, то надо вместо integer написать longint.
Вторая, мелкая разница, в том, что и real немного не тот, в нём не восемь байт, а только шесть, но тут уже разница во вместимости непринципиальна.
А вообще, между нами, программистами, говоря – в TP доступных типов данных сильно больше.

Глава 3.
Условные операторы

Что такое и зачем

Программа выполняется сверху вниз, это я уже говорил. А если надо не совсем сверху вниз, а так, зигзагом? А для этого у нас есть условные операторы.
Условный оператор – это очень просто. Покажем на примере. Программа определяет, положительное число, или отрицательное, и выдаёт соответствующее сообщение.

```
program Plus;
  uses
      Crt;
  var
      num  : integer;
begin
      Writeln( 'num = ');
      Readln(num);
      if num > 0 then Writeln('positive');
      if num < 0 then Writeln('negative');
end.
```

На интуитивном уровне вроде бы понятно. Теперь поподробнее – из чего состоит условный оператор.
Слово **IF** – в начале.
Дальше условие. Условие у нас пока что простое – два числовых выражения, разделенные знаком сравнения.
Знаки бывают вот такие:
> < - это понятно, больше и меньше
>= <= - больше или равно, меньше или равно
= - проверка на равенство
<> - а вот таким хитрым образом обозначается неравенство.
Дальше слово **THEN**.
А после него оператор. И оператор этот будет выполнен только тогда, когда выполняется условие, заданное нами (условие – это то, что между **IF** и **THEN**). Разумеется, оператор не должен обязательно быть

оператором вывода, а может быть любым – ну почти. Ну, и конечно, пробелы расставляются по вкусу, для достижения желаемого эстетического эффекта.
А теперь вспоминаем арифметический оператор **mod** и пишем программу проверки числа на чётность.
Кто быстро справился, вспоминает ещё и **div** и пишет программу проверки номера трамвайного шестизначного билета на счастливость – в смысле, чтобы сумма первых трёх цифр равнялась сумме трёх последних.

Усложняем

Два “а если”.
А если у нас не одно условие, а несколько – например, не просто больше нуля, а от нуля до ста?
А если у нас должен выполниться не один оператор, а несколько – например, сообщение о положительности числа покрасить в красный цвет, а об отрицательности – в синий?
Сначала ответ на второй вопрос
Вместо

```
if num>0 then Writeln(‘positive’);
```

пишем

```
if num>0 then begin
  TextColor(Red);
  Writeln(‘positive’);
end;
```

Появилась новая конструкция языка, которая в дальнейшем будет употребляться очень-очень часто.
Там где можно употребить один оператор, всегда (почти) можно употребить несколько, поставив перед первым **begin;**, а после последнего оператора **end;**.

По научному это называется операторные скобки.

Операторы, заключенные внутри пары **begin-end** могут быть любыми, в том числе условными операторами, содержащими **begin-end**, внутри которых…

И ещё. До сих пор у нас был только один **end** в конце программы, после которого стояла точка. После нашего нового **end**'а ставится точка с запятой. Это как раз типично, а **end** с точкой – аномалия.
Разумеется, наш исходный условный оператор

if num>0 **then** Writeln('positive');

можно записать и так

```
if num>0 then begin
  Writeln('positive');
end;
```

Кому что больше нравится.

А теперь что делать, если нам недостаточно простой проверки на положительность, а хочется чего-нибудь эдакого – например положительное, но не больше ста.
Важно! – если у нас условий несколько (два, например) каждое из них должно быть взято в скобки! В особенности это важно потому, что при отсутствии скобок сообщение компилятора об ошибке является неадекватным и вводящим в заблуждение, понять по нему, в чём дело трудно. Разумеется, у компилятора на это есть весьма уважительные причины.

Итак, имеем два условия:

(num>0) (num<=100).

Осталось объединить два простых условия в одно сложное:

(num>0) **and** (num<=100).

and обозначает, что мы требуем одновременного выполнения обоих условий.
В конечном итоге получаем:

```
if (num>0) and (num<=100) then begin
  Writeln('Ok');
end;
```

А какие ещё случаи бывают?
or - выполняется или первое условие, или второе, или оба сразу. Заметьте, это несколько отличается от бытового подхода – когда или-или, или одно или другое, но оба сразу – ни-ни. У нас можно и оба.

Not - отрицание. Эквивалентно замене условия на противоположное. То есть **not** (x>100)
то же самое, что
(x<=100).
Сложные условия также можно объединять между собой, получая условия, всё сложнее и сложнее.
Например (случай средней тяжести) - проверка, является ли год високосным:

```
(year mod 4 = 0) and ((year mod 100 <> 0) or ( (year div 100) mod 4 = 0)).
```

Если условие ещё сложнее, что-то в программе придумано не так…

Окончательно усложняем

А может, и упрощаем.
Пришёл к нам заказчик. И заказал программу проверки делимости числа на семь.
И сочинили мы вот такой шедевр:

```
program Divis;
  uses
     Crt;
  var
```