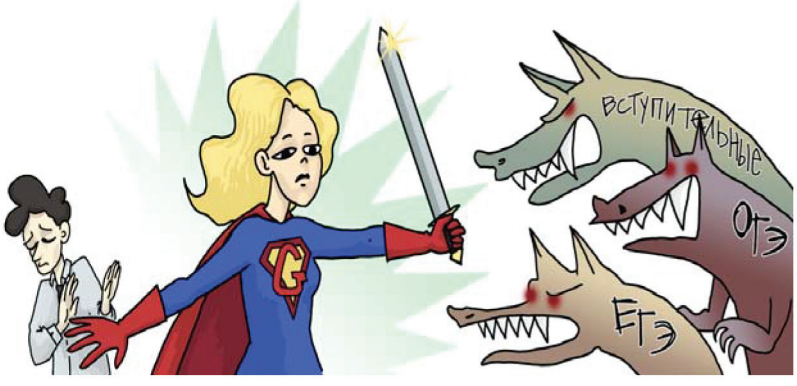
Информатика

Грацианова Татьяна Юрьевна
Кандидат физико-математических наук, преподаватель подготовительных курсов факультета Вычислительной математики и кибернетики МГУ им. М.В. Ломоносова.

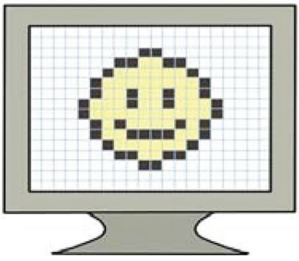
Графика, на помощь!
(Графика на службе ОГЭ и ЕГЭ)

Все задачки к экзамену уже разобраны, теперь надо только повторять, повторять, решать еще и еще. Да, скучно. Да, мозг устал после долгой зимы. Давайте взбодримся и откроем для себя стандартные понятия из программирования, информатики, математики с другой стороны.



Попробуем писать программы с рисунками, с графикой. Это умение не проверяется на экзамене, но в графических программах используется и ветвление, и циклы, и процедуры. А их наглядность не вызывает никаких сомнений.

Однако нечто новое (и даже не входящее в программу экзаменов) узнать всё же придётся: это набор простейших графических процедур (графических «примитивов») – без этого никак.



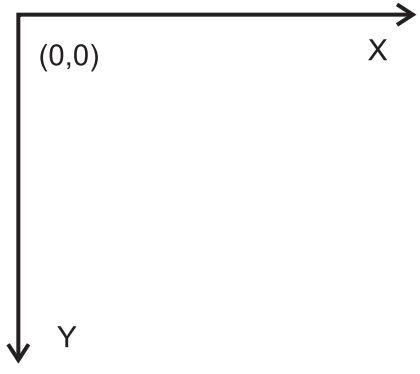
Чтобы программа создала рисунок, надо открыть специальный графический экран (или ещё говорят: перевести экран в графический режим). Дело в том, что в обычном, символьном режиме экран разделяется на позиции, знакоместа. В одной позиции может быть написан один символ. Экран в графическом режиме разделён на маленькие квадратики, фактически точки – пиксели. Каждый пиксель может быть своего цвета – благодаря этому и складывается картинка.

Будем экспериментировать в Паскале АВС (во FREE-Паскале тоже есть графическая библиотека, процедуры называются несколько по-другому, но алгоритмы будут те же).

Волшебное слово, открывающее графический экран выглядит так: **`uses GraphABC`**. Теперь то, что мы станем делать, будет печататься не внизу под текстом программы, а на этом новом экране. Возможно, вы эти слова уже и использовали в обычных программах – они будут работать так же и на графическом экране.

Как было сказано, рисунок – это набор цветных точек, поэтому, в общем-то, для рисования достаточно всего одной процедуры **`SetPixel(x,y,color)`**. У процедуры 3 параметра – координаты и цвет. Координаты на графическом экране выглядят несколько необычно: начало координат находится в верхнем левом углу экрана, ось X идет привычным образом: слева направо, а вот ось Y – иначе, сверху вниз. Почему так? Наверное, это «наследство» от выдачи информации в символьном виде: мы ведь читаем текст слева направо и сверху вниз.

Каков размер экрана? По умолчанию 640*400, при желании его можно менять – есть специальные процедуры.



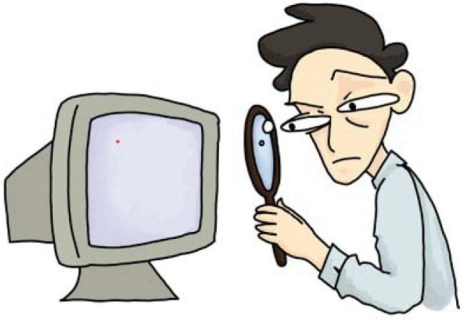
Теперь разберёмся с цветом. Для названий цветов существуют специальные константы. Мы не будем их все здесь перечислять, обычно имя такой константы состоит из букв cl (от слова color) и собственно названия цвета: clBlack, clRed.

Напишем нашу первую графическую программу:

```
uses GraphABC;
Begin
  SetPixel(10, 10,clRed)
End.
```

Нажимаем «Пуск». Новое окошечко видим, а больше ничего там нет.

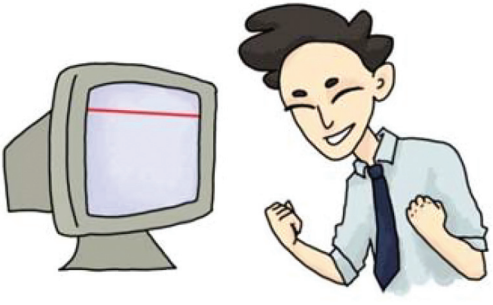
На самом деле, если присмотреться внимательно, нашу красненькую точечку разглядеть можно, но уж очень она маленькая. Сколько же таких точечек надо нарисовать, чтобы хоть что-то видимое (да еще и интересное) получить!



Но у нас есть возможность использовать оператор цикла! Напишем процедуру установки точки один раз, но сделаем это в цикле – получим много точек. Вот так:

```
uses GraphABC;
var X : Integer;
Begin
  For X:=0 to 640 do
     SetPixel(X, 10,clRed)
End.
```

Почему от 1 до 640? Таков размер экрана. Координата X сначала равна 0 – нарисуется точка (0, 10). Потом (вспоминаем, как работает цикл!) X станет равно 1, нарисуется точка (1, 10) и так далее до (640, 10).



А теперь получим несколько линий. Для этого надо менять Y, причём во внешнем цикле (Y надо описать в разделе описаний рядом с X):

```
For Y:=0 to 400 do
  For X:=0 to 640 do
     SetPixel(X, Y,clRed)
```

И что мы видим? Никаких прямых линий нет: экран у нас на глазах равномерно окрашивается в красный цвет, пока не станет

весь красным. В чём дело? Так ведь это и есть наши линии, только они нарисованы рядом, без промежутков, вот и сливаются в один прямоугольник.

Надо линии разделить, т.е. Y не должен меняться по формуле Y:=Y+1, а, например, по формуле Y:=Y+10. Но в этом случае переменную Y нельзя использовать как переменную цикла: ведь в цикле FOR шаг её изменения всегда 1. В качестве параметра цикла возьмём другую переменную, Y будем использовать в процедуре и не забудем, что теперь мы его менять должны сами. Это надо сделать внутри цикла. Запишем там Y:=Y+10. В цикле оказалось 2 оператора, заключим их в операторные скобки Begin – End. И установим начальное значение Y (пусть оно будет равно 5).

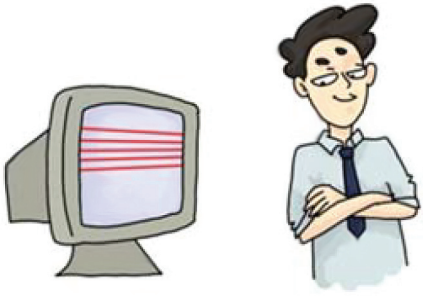
Программа очень простая, но здесь важно не перепутать циклы. Во внутреннем цикле (по X) у нас по точкам рисуется горизонтальная прямая линия. Во внешнем цикле (по Y) происходит 2 действия: рисование линии (оператор цикла) и изменение значения Y. В операторные скобки надо заключить именно эти операторы.

А как закончить внешний цикл? Раньше у нас там стояло число 400, так как мы, закрашивая весь экран, рисовали 400 отрезков. Теперь нам столько не нужно, да и не поместятся они на экране – ведь теперь между ними будет расстояние. Нарисуем 15 линий:

```
Y:=5;
 For J:=1 to 15 do
 Begin
    For X:=1 to 640 do
         SetPixel(X, Y,clRed);
    Y:=Y+10
 End
```

Мы нарисовали 15 линий, оказалась заполненной часть экрана. А как заполнить такими линиями весь экран? Для этого придётся немножко посчитать. На экране по вертикали 400 точек, расстояния между прямыми – 10, да ещё первая прямая находится не на самом верху, а на 5 ниже (такое начальное значение нами установлено). (400-5) делим на 10, используем целочисленное деление, ведь прямых будет целое число. Получаем 39. Столько будет промежутков между линиями. А линий, значит, надо нарисовать 40.

Проверим себя двумя способами: запустим программу с заго-



ловком цикла For J:=1 to 40 do (линии должны заполнить всё поле) и посчитаем последнее значение Y (кстати, такие задачки есть на эк замене – требуется определить, чему будет равно значение переменной). Посмотрим внимательно, как работает внешний цикл

До цикла Y=5.
Шаг цикла J=1, рисуется прямая на высоте 5, изменяется Y=15.
Шаг цикла J=2, рисуется прямая на высоте 15, изменяется Y=25.
Шаг цикла j – общая формула: прямая на высоте 5+(J-1)*10, Y становится 5+J*10.
Шаг цикла J = 40 – прямая на высоте 395, Y = 405.

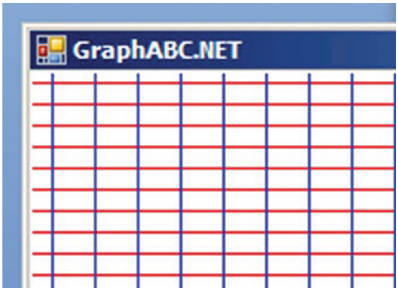
Да, последнее значение выходит за границы экрана, но мы ведь и не рисуем прямую на такой высоте, последний Y нами не используется. А правильность расчётов можно проверить и ещё одним способом – просто после цикла напечатать значение Y.

А если считать лень? У нас ведь компьютер – пусть он считает. Напишем такую же программу, воспользовавшись другим оператором цикла – Repeat. Заменим заголовок цикла FOR на служебное слово REPEAT, уберём операторные скобки, (в этом цикле они не нужны). В конце напишем условие выхода из цикла: закончить цикл, как только высота выйдет за пределы экрана, т.е. станет больше 400 – пишем Until Y>400. Программа будет работать точно так же.

А теперь можно сделать еще одну картинку: нарисовать не горизонтальные, а вертикальные линии. Программа для этого будет выглядеть точно так же, только координаты поменяются местами, и надо учесть, что размеры экрана по горизонтали другие.

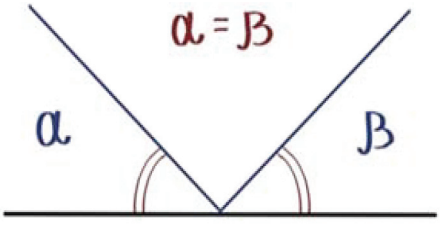
```
uses GraphABC;
var X, Y, J : Integer;
Begin
 Y:=5;
 Repeat
     For X:=1 to 640 do
         SetPixel(X, Y,clRed);
    Y:=Y+10
 Until Y>400;
 X:=10;
 Repeat
     For Y:=1 to 400 do
         SetPixel(X, Y,clBlue);
    X:=X+20
 Until X>640;
End.
```

Если всё сделано правильно, должна получиться картинка, фрагмент которой изображён на рисунке.



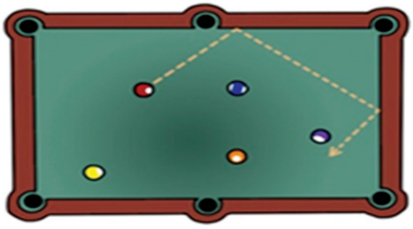
А теперь давайте вспомним физику. Смоделируем движение шарика по бильярдному столу: наш экран будет столом, а точка – пиксель –

шариком. Он будет катиться по «столу» до одной из границ, отталкиваться от неё и катиться в обратную сторону. По какому закону он будет отталкиваться? Угол падения равен углу отражения. Правда, это про свет учили, но с шариком произойдёт то же самое.

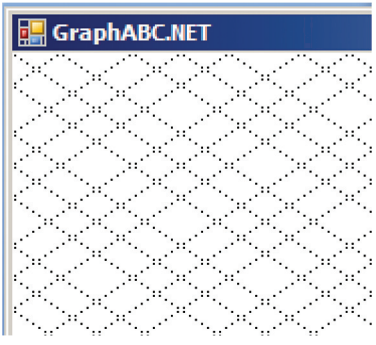


Шарик будем рисовать с помощью процедуры SetPixel., начальные координаты зададим в начале программы. А как шарик будет двигаться? Станем изменять его координаты и заново рисовать. Для совсем правильной имитации движения надо было бы ещё стереть предыдущее изображение – именно так рисуют мультфильмы, но мы этого делать не станем. Будем считать, что шарик двигается по экрану, оставляя за собой след. Менять координаты будем, прибавляя к ним некоторое «приращение» – небольшое число. Обозначим его по горизонтали dX, по вертикали dY. Установим начальные значения этих переменных. Их значения лучше выбрать небольшими (1 – 5), чтобы шарик не слишком резко перескакивал с одного места на другое. От значений переменных зависит, под каким углом будет двигаться шарик. Скажем, если одно из значений сделать равным 0, шарик станет двигаться только по горизонтали или по вертикали – мы ничего интересного не увидим. Если приращения сделать одинаковыми, шарик покатится под углом 45° к границе экрана.

Для движения шарика будем менять его координаты, прибавляя к ним приращения. Если изменить знак приращения, сделать dX:=-dX (не прибавлять, а отнимать), то шарик покатится в обратную сторону. Это и есть реализация правила «угол падения равен углу отражения».



Конечно же, нам понадобится оператор цикла. Какой – зависит от того, как мы хотим закончить движение. Выберем пока самый простой случай – изобразим 100000 точек, т.е. воспользуемся циклом FOR.

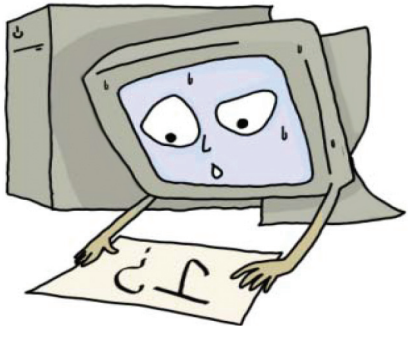


Что будет внутри цикла? Сначала поставим точку с координатами (X, Y). Теперь меняем координаты. Начнём с X, прибавив приращение. Готово? Нет! А вдруг шарик был вблизи границы экрана и после из-

менения значение X уже выходит за границы? Точку с такими неправильными координатами изображать нельзя! Границы экрана по горизонтали мы знаем. Это 0 и 640. Надо проверить, не «ускачет» ли наш шарик за экран. Эта неприятность случится, если (X<0)OR (X>640). Обратите внимание: два условия связываем оператором ИЛИ. – Если выполнится ОДНО из неравенств, шарик окажется за экраном (а одновременно они выполниться никогда и не могут).

Что же делать, если шарик на следующем шаге может оказаться за экраном? Менять приращение, вернее, менять его знак на противоположный. При этом не важно, убегает шарик вправо или влево: в одном случае знак меняется с «+» на «-», в другом с «-» на «+», а формула одна: dX:= –dX. Кроме этого, надо ещё и координату X сделать прежней.

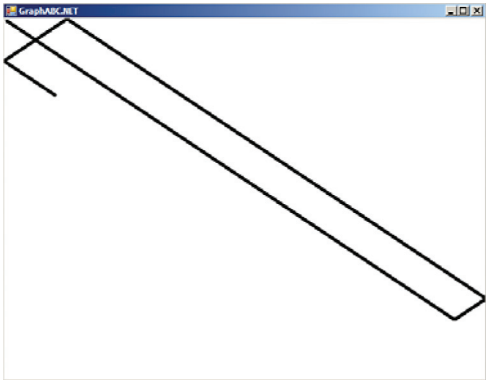
Точно такой же фрагмент пишем для координаты Y, учитывая, что по вертикали у экрана другой размер.



Запускаем. На экране мельтешат точки, и очень скоро появляется картинка – фрагмент которой изображён ниже. А хотелось бы наблюдать, как наш шарик катается, как он эту картинку рисует. Почему этого не видно? Да потому, что наш компьютер слишком быстро работает. Замедлим его немножко.

Для этого существуют специальные средства, но мы можем обойтись и тем, что знаем. Вставим в конце нашего цикла оператор, который будет «тянуть время»:

```
For J:= -1000000 to 1000000 do;
```



Как видите, это оператор цикла, который ничего не делает (после do стоит точка с запятой, т.е. тело цикла – пустой оператор). Впрочем, говорить, что он совсем уж ничего не делает, неправильно. Ведь он считает, изменяет свою переменную цикла J, причём делает это 2000001 раз. Какую-то долю секунды компьютер на это потратит, что замедлит бег шарика и даст нам возможность увидеть через пару секунд после запуска программы, как он перемещается.

Приведём полную программу – вдруг у кого-то что-то пошло не так (кстати, если рисунок получается некрасивый, можно попробовать поменять начальные значения положения точки и приращение):

```
uses GraphABC;
var X, Y,J, dX, DY, I : Integer;
Begin
  X:=1; Y:=1;
  dX:=3; dY:=2;
  For I:=1 to 9999 do
  Begin
     SetPixel(X, Y,  ClBlack);
     X:=X+dX;
     if (X>640) OR (X<0) then Begin  X:=X-dX;
                                     dX:=-dX
                              End;
     Y:=Y+dY;
     if (Y>400) OR (Y<0) then Begin  Y:=Y-dY;
                                     dY:=-dY
                              End;
     SetPixel(X, Y,  ClBlack);
     For J:=-1000000 to 1000000 do;
  End
End.
```

Вот какие интересные вещи можно запрограммировать, используя всего лишь один графический «примитив». А ведь есть и другие графические процедуры. Но об этом в следующий раз!