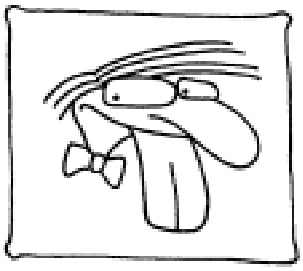
Романовский Иосиф Владимирович

ЗАНЯТИЕ 4.
ЯЗЫК ПРОГРАММИРОВАНИЯ ПОСТСКРИПТ

Введение

В этой статье я хочу познакомить вас с одним интересным языком программирования, который сильно отличается от языков традиционной линии Паскаль-Си-Java.



Это язык PostScript, алгоритмический язык, предназначенный для описания изображений. Он был разработан в 1985 году американской фирмой Adobe Systems, Inc. для того, чтобы упростить передачу информации из компьютера в принтер.

Дело в том, что с ростом разрешения принтеров, измеряемого количеством точек на единицу длины, объем информации о странице растет пропорционально квадрату этого разрешения. Возникла идея - разработать алгоритмический язык, чтобы компьютер описывал на этом языке печатаемую страницу, а находящийся в принтере интерпретатор языка выполнял бы эту программу и тем самым рисовал страницу. Так и было сделано. Были разработаны специальные «постскриптовские» принтеры и даже высокоточные наборные устройства, работающие с этим языком.

В дальнейшем обнаружилось, что иногда (даже часто) хочется до печати изображения на бумаге или наборной пленке посмотреть, что получается, на экране дисплея. С этой целью были созданы интерпретаторы Постскрипта, работающие в обычном компьютере. Эти интерпретаторы оказались очень удобны; они позволяют не только показывать результат на экране, но и печатать его практически на любом принтере (вплоть до самого дешевого 9-точечного), а не только на специальном. Более того, они позволяют выводить результаты в виде графического файла в одном из стандартных форматов. Именно так мы и подготовили иллюстрации к этой статье.

Некоторые общие элементы языка

Сначала немного об общих принципах языка. В его основе лежит понятие стека, которое, я надеюсь, вам знакомо. *Стек* - это универсальное хранилище данных, чисел, строк, массивов, процедур. Каждый элемент данных занимает в стеке одно место, все места одинакового размера. Помещаемый в стек элемент располагается сверху.

Кроме Постскрипта существуют и другие стековые языки. Из них наиболее известен Форт (Forth), но, пожалуй, Постскрипт сейчас наиболее распространен, и он дает самую удобную возможность попробовать эту увлекательную технику программирования.

В Постскрипте есть некоторое (довольно большое) число стандартных слов, которые можно назвать командами, действиями, операторами, функциями или процедурами языка. Мы будем пользоваться этими названиями как синонимами, так как, например, слово **`add`**, описывающее сложение, нам естественно назвать операцией, а слово **`sin`**, вычисляющее синус, - функцией, а установку ширины линии - действием или командой.

Основные действия производятся с элементами стека. Например, программный текст

```
32 0.5 43 17 add mul sub
```

исполняется так: положить в стек число 32, выше 0.5, затем 43, затем 17, после этого выполнить действие **add**, затем **mul**, затем **sub**. Действие **add** - это сложение. Оно снимает со стека два числа, складывает их и кладет результат в стек. После этого действия в стеке будут лежать

```
32 0.5 60
```

Действие **mul** - это умножение. Аналогично сложению оно перемножает два снятых со стека числа, а результат кладет в стек. Действие **sub** - вычитание, верхнее число вычитается из нижнего, так что в нашем примере окончательно получится 2. Вполне аналогично этим действиям определяются деление **div**, деление нацело **idiv**, остаток от деления **mod**, а также одноместные операции изменения знака **neg** и нахождения абсолютного значения **abs**.

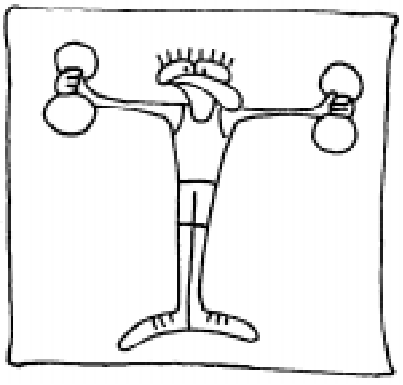
Имеются операции вычисления некоторых стандартных функций (не всех):

exp	Вычисление экспоненты
log	Десятичный логарифм
ln	Натуральный логарифм
sqrt	Квадратный корень
cos	Косинус угла, аргумент задается в градусах
sin	Синус угла
atan	Арктангенс, его аргумент задается как пара (числитель, знаменатель)

Табличная форма описания кажется очень удобной, и следующую группу действий мы опишем прямо таблицей. Это манипуляции со стеком (в первом столбце название действия, во втором объяснение, в третьем и четвертом - пример данных в стеке до и после выполнения действия)

dup	Копирует вершину стека	31 75 43	31 75 43 43
pop	Удаляет вершину стека	31 75 43	31 75
exch	Переставляет два верхних элемента	31 75 43	31 43 75
roll	Снимает со стека два параметра, **m** и **n**, в оставшейся части стека выделяет группу из **n** элементов и верхние **m** из них опускает вниз	31 75 43 12 61 5 2	12 61 31 75 43
index	Снимает со стека параметр **n** и копирует элемент, находящийся на месте **n**, считая вершину нулевым элементом	31 75 43 12 61 2	31 75 43 12 61 43

Сделаем маленькое упражнение. Вычислим по паре чисел корень из суммы их квадратов - очень важное и знакомое нам действие, это вычисление гипотенузы по катетам. В следующем ниже постскриптовском тексте есть примечания; примечание начинается знаком процента и идет до конца строки. В каждой строке этого упражнения приводится состояние стека после выполнения действия:



```
        %   a b    это исходное состояние стека (интересующей нас части)
dup     %   a b b
mul     %   a bXb
exch    %   bXb a
dup     %   bXb a a
mul     %   bXb aXa
add     %   bXb+aXa
sqrt    %     желанный результат
```

Очень важно, что такие действия, которые мы описываем сами, можно превращать в слова, которые потом, наряду со стандартными словами, используются для вычислений и для описания других слов.

Описание слова состоит из его названия со знаком косой черты, действий, составляющих слово, и завершающего слова **def**. И название и действия должны занимать по одной позиции стека, а если мы хотим, чтобы слово вызывало выполнение нескольких операций, этот набор операций должен быть помещен в фигурные скобки. Таким образом, описанное нами действие можно оформить как новое слово (комментарии теперь можно убрать)

```
/hypot{ dup mul exch dup mul add sqrt } def
```

А схожее описание

```
/h a b dup mul exch dup mul add sqrt def
```

определит константу **h** по значениям **a** и **b**, выработанным в момент исполнения этого описания. Слово **h** будет класть эту константу в стек. Мы могли бы определить **h** и проще, с использованием слова **hypot**

```
/h a b hypot def
```

Возможность добавления описаний просто замечательна, она используется и для введения констант и переменных. Да, и переменных, так как слова можно переопределять. Например, вот так выглядит прибавление единицы к счетчику **iter** и увеличение накопителя **sum** на величину, лежащую в стеке

```
/iter iter 1 add def
   % в стек кладется название слова, затем связанное с ним
   % значение, затем значение увеличивается и оставляется
   % в стеке, затем это новое значение связывается с данным
   % словом.
/sum exch sum add def
   % в стеке уже лежит значение, добавляемое к sum, его
   % нужно поменять местами с названием, остальное как выше.
```

Обратите внимание на то, что значение, лежащее в стеке, и название можно просто переставить, в стеке они равноправны.

В Постскрипте есть возможности условного исполнения и организации циклов, для удобства имеются логические переменные и операции над ними. Мы не будем тратить на них место. Следующее, что нас должно интересовать, - это специфические действия и объекты, связанные с изображениями.

Геометрические объекты и действия с ними

Постскрипт манипулирует с таким привычным и школьным объектом, как евклидова плоскость с введенной на ней системой самых обычных декартовых координат. Представьте себе просто лист бумаги. Первоначально начало координат находится в его левом нижнем углу, ось X идет вправо, ось Y вверх, единицей измерения является пункт (point), 1/72 часть дюйма[1]. Углы измеряются в градусах и отсчитываются от оси X против часовой стрелки.

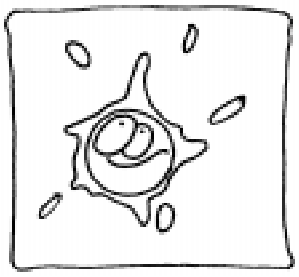
Начало координат можно изменить, для этого нужно положить в стек координаты нового начала, вычисленные в текущей координатной системе, и выполнить действие **translate**, которое снимет эти координаты со стека и изменит систему. Можно изменить шкалу единиц отдельно по оси X и по оси Y, для этого служит действие **scale**. Отметим, что

```
-1 1 scale
```

выполняет зеркальное отображение плоскости. А можно повернуть систему координат:

```
35 rotate
```

повернет координатные оси против часовой стрелки на 35°.



Есть понятие *текущей точки* - это такая особая системная переменная, которая может быть и не установлена. Ее установка выполняется действием **moveto**, которое снимает со стека два числа и трактует их как новые координаты текущей точки. Например,

```
300 150 moveto
```

Слово **currentpoint** кладет в стек координаты текущей точки. Наряду с **moveto**, работающей с абсолютными координатами точки, имеется действие **rmoveto**, добавляющее к координатам текущей точки требуемые значения. Таким образом, для того, чтобы сдвинуть текущую точку вверх на 120, нужно выполнить

```
0 120 rmoveto
```

Когда мы хотим что-то нарисовать, а Постскрипт предназначен именно для рисования, и все, о чем мы пока говорили, носит вспомогательный характер, мы должны создать рисовальный контур, который называется *путем*. Так сложно! Не прямо рисуем линию, а сначала создаем контур! Но это усложнение имеет смысл: созданный контур можно потом использовать для разных целей.

Самый простой путь состоит из прямолинейных участков. Действие **lineto** проводит отрезок от текущей точки до точки, координаты которой берутся со стека.

---

[1] Странно, да? Дело в том, что Постскрипт язык полиграфический, а в полиграфии система мер была введена в конце XVIII века французскими полиграфистами Дидо. Их система мер так и используется, только прогрессивные американцы используют свой новый дюйм, а мы - старый французский. (Зато мы давно перешли на метрическую систему, а эти прогрессивные американцы все еще собираются, заставляя весь «метрический» мир измерять в дюймах размеры компьютерных деталей). Обычные книги набираются кеглем 10 пунктов, а школьные учебники - 12 пунктов (кегль это не высота буквы, а высота типографской литеры, примерно, это расстояние между нижними кромками двух соседних строк).

Эта концевая точка становится текущей и, конечно может быть использована дальше. Например,

```
300 250 moveto
350 250 lineto 350 220 lineto
400 220 lineto 400 190 lineto
450 190 lineto 450 160 lineto
500 160 lineto 500 130 lineto
```

- получилась лесенка из четырех ступенек. У **lineto**, как вы догадываетесь, есть брат, работающий с приращениями, и его, конечно, зовут **rlineto**. Смотрите, как просто пишется то же самое с его помощью

```
300 250 moveto
50 0 rlineto 0 -30 rlineto
50 0 rlineto 0 -30 rlineto
50 0 rlineto 0 -30 rlineto
50 0 rlineto 0 -30 rlineto
```

Вот бы нам цикл устроить! Написать просто, сколько раз повторить такое действие. Действительно, такой цикл есть. Он выполняется командой **repeat**, перед которой пишется число повторов и в фигурных скобках повторяемое действие. Вот так

```
300 250 moveto
4 { 50 0 rlineto 0 -30 rlineto } repeat
```

Теперь можно накопленный путь нарисовать. Для этого есть действие **stroke**.

При исполнении **stroke** путь и текущая точка теряются, и нужно все начинать заново. Не будем расстраиваться, выход найдется. Пока поговорим о том, как много у нас есть вариантов рисования нашей линии.

Прежде всего, можно установить ширину линии. Ширина также измеряется в пунктах, но при этом, если вы измените масштаб, то ширина изменится. Для установки используется команда **setlinewidth**, например,

```
0.5 setlinewidth
```

Можно изменить «пунктирность» линии (по умолчанию линия, конечно, сплошная), способ стыковки прямоугольных участков, способ оформления свободного конца, - мы всего описать не сможем.

Очень важна возможность менять цвет изображения. Цвет можно выбирать как градацию серого (это очень удобно, если у вас черно-белый принтер, потому что он сможет изобразить серый цвет любой яркости, смешивая черные и белые точки в нужной пропорции). Установка этой градации выполняется действием

```
a setgray
```

где **a** задает этот уровень, 0 для сплошного черного, 1 для сплошного белого, чем ближе к 1, тем светлее.

Другая возможность - выбор полноценного цвета. В Постскрипте используется система описания цветов RGB, в которой цвет получается смешением красного (Red),

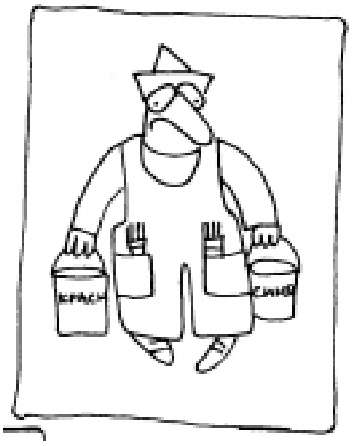
зеленого (Green) и синего компонентов (Blue). Доля каждого компонента лежит между 0 и 1. Наиболее удобные цвета можно заранее описать как команды. Например,

```
/RED{ 1 0 0 setrgbcolor}def          %  красный
/WHITE{ 1 1 1 setrgbcolor}def        %  белый
/YELLOW{ 1 1 0 setrgbcolor}def       %  желтый
/BROWN{0.5 0.5 0 setrgbcolor}def     % коричневый
```

Можно с помощью Постскрипта хорошо поэкспериментировать с разными цветами. Отметим, что постскриптовские краски «непрозрачные», что было под краской, уже не видно.

Мало того, что мы можем нарисовать построенный путь. В случае, если этот путь замкнут, можно закрасить его внутренность (залить ее) установленным в настоящее время цветом. Заливку выполняет команда **fill**. Как и **stroke**, эта команда уничтожает использованный ею путь и установку текущей точки. Как странно, что же делать, если мы хотим нарисовать, например, желтую фигуру с зеленой рамкой? Должен же быть какой-то выход. Выход есть.

Стек графических состояний



Вся совокупность установок, влияющих на рисование объектов, - цвет, толщина и другие параметры линии, начало координат, масштабы и направление координатных осей, - все это образует графическое состояние.

Имеется специальное хранилище графических состояний, конечно же, тоже стек; и текущее графическое состояние можно сохранить в стеке командой **gsave**, а затем восстановить командой **grestore**.

Таким образом, чтобы нарисовать желтую фигуру с зеленой рамкой, нужно выполнить

```
gsave YELLOW fill grestore GREEN stroke
```

то есть сохранить графическое состояние, установить желтый цвет, залить им фигуру, восстановить графическое состояние (а значит, и путь и цвет) и прорисовать контур.

Невозможный треугольник

Рассмотрим пример, в котором все, что мы уже узнали, используется для рисования красивой картинки. Это «Невозможный треугольник» Пенрозов, использованный замечательным голландским художником М. Эшером для своей литографии «Водопад» (Рисунок 1).

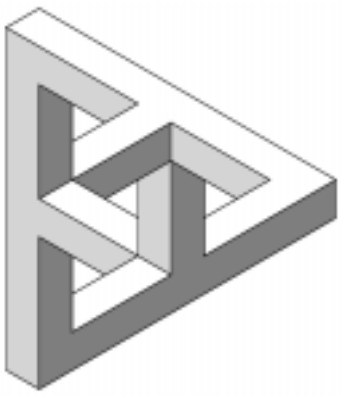


Рисунок 1.

Полный текст на Постскрипте:

```
%!   Начало файла
% Некоторые определения
/l{lineto}def /S{stroke}def   % Сокращения стандартных слов%!
/FS{gsave setgray fill grestore S}def
   % закраска и обвод. Обратите внимание:
   % уровень «серости» берется из стека

/Triangle{  P1 0.5 FS  M P1 0.8 FS M P1 S}def
   % Фигура состоит из трех одинаковых частей, они
   % выкрашены в серый, светлосерый и белый цвет и
   % повернуты друг относительно друга на 120 градусов.
   % Рисование контура выполняет P1, а поворот и
   % перенос начальной точки - M
   % За начальную я принял нижнюю точку картинки,
   % поэтому начальная точка светлосерой части
   % отстоит на один шаг влево и девять шагов вверх

/M{p1 1 L 9 U translate -120 rotate} def
   % p1 - начальная точка (см. ниже)
   % новые команды L и U выполняют шаги влево и вверх
   % наряду с ними будет и команда R для шагов вправо
/U { step mul add} def
   % шаги вверх самые простые: умножаем число шагов
   % на размер шага, определяющий масштаб изображения,
   % и прибавляем результат к координате Y точки,
   % лежащей в стеке.
   % Для дальнейшего потребуется команда, которая
   % к точке (x0,y0) прибавляет (a*x1,a*y1). Спокойно
   % напишем эту команду, фиксируя состояние стека
   % после каждого действия
/offset {       % x0 y0 a x1 y1 - начальное состояние
    2 index     % x0 y0 a x1 y1 a
    mul         % x0 y0 a x1 a*y1
    3 1 roll    % x0 y0 a*y1 a x1
    mul         % x0 y0 a*y1 a*x1
    4 3 roll    % y0 a*y1 a*x1 x0
    add         % y0 a*y1 x'=x0+a*x1
    3 1 roll add } def
   % Имея команду offset, написать шаги влево и вправо
   % уже нетрудно, нужно только запасти одну константу
   % - косинус 30 градусов
/cos30 0.75 sqrt def
/L {step mul cos30 neg 0.5 offset}def
/R {step mul cos30 0.5 offset}def
   % С командой M покончено. Параметр step остается
   % не определенным, - оставим на «потом».

   % Теперь P1: если вы присмотритесь, то увидите, что
   % все три куска рисуются как один контур
/P1{ p1 moveto
```

```
      p2 l p6 l p4 l p8 l p13 l p16 l p17 l p18 l
      p14 l p15 l p9 l p11 l p10 l p12 l p7 l p1 l} def
    % соответствующие точки выражаются друг через друга
/p1 {0 0} def  % начало
/p2 {p1 4 U} def /p16 {p1 5 U}def % вверх от начала
/p17 {p1 8 U} def
/p15 {p2 2 R}def                  % вправо от p2
/p3 {p1 1 R} def /p7 {p1 9 R} def % вправо от начала
/p4 {p3 1 U} def /p5 {p3 2 U} def % вверх от p3
/p6 {p3 3 U} def /p14 {p3 5 U}def /p18 {p3 7 U} def
/p8 {p4 3 R} def /p10 {p4 4 R}def % вправо от p4
/p12 {p4 8 R}def
/p9 {p8 3 U} def /p13 {p8 4 U}def % вверх от p8
/p11 {p10 2 U}def                 % вверх от p10

    % осталось выбрать шаг, сдвинуть начало координат
    % и нарисовать
/step 40 def
200 200 translate
Triangle
Showpage % завершающее действие, которое позволяет
 % принтеру вывести готовую страницу
```

Кривые линии

Конечно же, хочется рисовать не только прямые линии, но и..., разумеется, например, окружности. Команда **arc** прорисовывает (добавляет к пути) дугу окружности. Ее точный формат

```
x y r a1 a2 arc
```

при таких параметрах проводится дуга окружности с центром в точке (x,y) и радиусом r от угла a1 до угла a2. Дуга проводится против часовой стрелки, а для дуг, проводимых по часовой стрелке имеется команда **arcn**. Уточнение: до рисования дуги проводится отрезок от текущей точки до начала дуги, если вы не хотите, чтобы этот отрезок появился, установите сами правильную начальную точку.

Вот красивый пример - старинный восточный символ Инянь, олицетворяющий соединение двух противоположных начал (Рисунок 2).

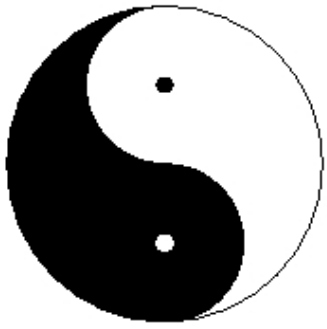


Рисунок 2.

Текст программы совсем короткий

```
%!
/m{moveto}def  /F{fill}def
  % пара удобных сокращений
/disk {dup currentpoint 4 2 roll 0 rmoveto 0 360 arc}def
  % полезное действие, рисует окружность данного
  % радиуса с центром в текущей точке

/R 20 def /RH R 0.5 mul def /Rhole R 0.05 mul def
```

```
300 500 translate
   % черная часть фигуры
0 R m 0 0 R 90 270 arc % левая часть
0 RH neg RH 270 90 arc % низ правой части
0 RH dup 270 90 arcn F % верх правой части
   % круглые точки
0 RH     m Rhole disk F % черная
gsave
0 RH neg m Rhole disk 1 setgray F
grestore                % белая
   % внешняя окружность
0 setlinewidth
0 0 m R disk stroke
showpage
```

Кроме непосредственного рисования дуг окружностей, в Постскрипте предусмотрена возможность рисования *закруглений углов*. Действия **arcto** и **arct** вписывают дугу данного радиуса в заданный угол. Мы здесь рассмотрим только второе из этих действий. Его формат

```
x1 y1 x2 y2 r arct
```

Пусть текущая точка обозначена через A, и имеются точки B=(x1,y1) и C=(x2,y2). В угол ∠ABC вписывается окружность радиуса r. Действие **arct** проводит прямую линию по AB до точки касания и дугу окружности до точки касания на BC. Эта вторая точка касания и становится текущей.

Еще одна замечательная возможность, о которой мы только упомянем, - это проведение сглаживающих кривых (они называются кривыми Безье[2]). Действие

```
x1 y1 x2 y2 x3 y3 curveto
```

проводит гладкую кривую, соединяющую текущую точку (x0,y0) с точкой (x3,y3), причем направление этой кривой в начальной точке совпадает с направлением на точку (x1,y1), а в конечной точке - с направлением на (x2,y2). Степень прилегания кривой к направлениям зависит от удаленности этих направляющих точек. (Для тех, кто жаждет математических подробностей: кривая Безье задается параметрически полиномами третей степени

$x(t) = a_x t^3 + b_x t^2 + c_x + x_0$,

$y(t) = a_y t^3 + b_y t^2 + c_y + y_0$,

причем параметр t пробегает значения от 0 до 1. Контрольные точки связаны с коэффициентами следующими равенствами

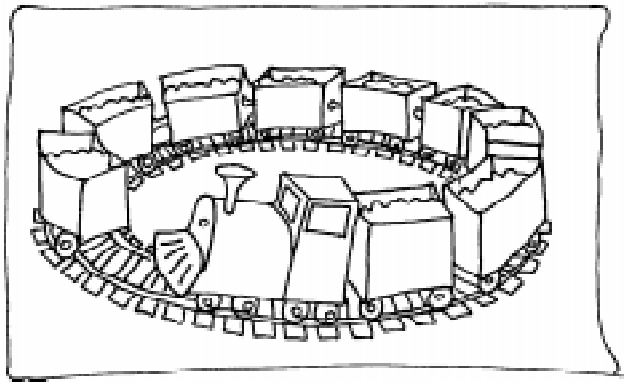
$x1 = x_0 + c_x/3$, $x2 = x1 + (c_x + b_x)/3$, $x3 = x_0 + a_x + b_x + c_x$,

и совершенно аналогично для y.)

Рисование внутри контура

Сейчас настал момент описать еще одно использование замкнутых путей. Путь может задавать область, внутри которой должна находиться видимая часть рисунка.

[2] Пьер Этьен Безье, французский инженер-автомобилист, придумал эту кривую в 1970-е годы. Д. Кнут, который использует кривые Безье в своей системе METAFONT, отмечает, что кривая Безье является частным случаем знаменитых полиномов Бернштейна.



Для того, чтобы достичь такого эффекта, нужно после построения замкнутого контура выполнить действие **clip**, а затем действие **newpath**, открывающее образование нового пути. Рисование этого нового пути (или закраска) использует ограничивающий контур и вызовет его потерю. Однако, так как ограничивающий контур входит в графическое состояние, то сохранением состояния, можно сохранить и его. Рассмотрим небольшой пример (Рисунок 3).

Посмотрим, что нужно сделать, чтобы несколько геометрических фигур «подсунуть друг под друга» в циклическом порядке, как это показано на центральном рисунке. Если мы будем просто рисовать одну фигуру за другой, то последняя нарисованная фигура окажется верхней. На левом рисунке показано, что получилось, причем тонкой рамкой обведено неправильное место. Его очень просто исправить, если внутри рамки нарисовать заново первую фигуру. Так и получен центральный рисунок. Посмотрим, как были сделаны эти два рисунка. А правый рисунок показывает, что тем же способом можно делать и более сложные вещи. Итак

Рисунок 3.

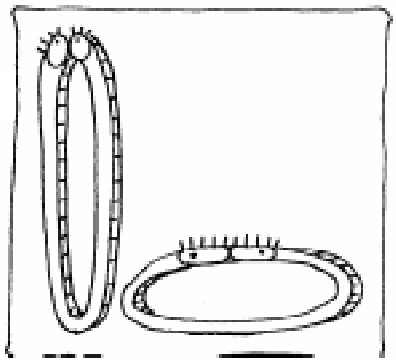
```
%!
     % Некоторые удобные действия
/m{moveto}def  /l{lineto}def  /rl{rlineto}def
/rm{rmoveto}def /hrl{0 rl}def /hm{0 m}def /vrl{0 exch rl}def
/vm{0 exch m}def /hrm{0 rm}def /vrm{0 exch rm}def
/bp{0 0 m}def /CL{closepath}def /CPT{currentpoint translate}def
/GS  {gsave}def  /GR{grestore}def

/FS{GS 0.75 setgray fill GR stroke}def
   % клин
/V { 30 -60 rm -7.5 90 rl 15 hrl CL FS} def
   % Левая картинка
1.2 setlinewidth
   % циклом рисуются шесть закрашенных клиньев
120 340 m 6 {GS CPT V GR 60 rotate} repeat
   % рамка, которая обводит «неправильное»
   % место, где самый последний клин лежит
   % выше всех
GS 21 -31 rm 25 vrl 25 hrl -25 vrl CL
0.5 setlinewidth stroke GR
   % Центральная картинка
   % сначала такой же цикл из 6 клиньев
260 340 m 6 {GS CPT V GR 60 rotate} repeat
```

```
   % потом строится та же рамка
GS 21 -31 rm 25 vrl 25 hrl -25 vrl CL
   % дальше эта рамка не рисуется, а объявляется
   % ограничивающим контуром, и путь начинает
   % строиться заново
clip  newpath 260 340 m V GR
   % нарисованный клин изобразился только
   % внутри рамки
   % Правая картинка
   % Место для программы
   %
showpage

quit
```

Тексты и шрифты



В своем важнейшем использовании - для подготовки печатных документов Постскрипт не может обойтись без вывода текстов.

Вывод осуществляется очень просто: команда `show` снимает со стека текстовую строку (строка, независимо от размера, занимает в стеке одну позицию) и изображает ее установленным шрифтом, размером и цветом, начиная с текущей точки. Строка может быть задана непосредственно, она ограничивается обычными круглыми скобками. Парные скобки внутри строки воспринимаются правильно, если нужна непарная скобка, то ей должна предшествовать обратная косая черта:

```
(И, наконец, 3\)  О текстах и шрифтах) show
```

Есть несколько вариантов команды **show**, позволяющих различным образом изменять расстояние между буквами, на них у нас, к сожалению, нет места. Но невозможно умолчать о команде **charpath**, которая формирует контур строки и добавляет его к имеющемуся пути. Дополнительный логический параметр со значением **«ДЛЯ stroke»** определяет назначение этого контура.

Что же касается установки шрифтов, то в первоначальном Постскрипте проблема была решена очень просто: шрифт определялся как набор очень экономно написанных процедур рисования отдельных букв, изображенных в одном и том же масштабе. Каждый такой готовый шрифт записывается в отдельном файле (вместе с некоторой справочной информацией), он получает уникальное имя, по которому может быть найден. Найденный шрифт может быть «шкалирован» - приведен к требуемому масштабу и установлен в качестве текущего шрифта.

Похвастаюсь шрифтом, который я спроектировал сам, используя образцы начала века и замечательную программную систему **FontLab**[3] .

[3] Пользуюсь случаем, чтобы рекомендовать вашему вниманию и саму эту систему, разработанную в Петербурге и считающуюся одной из лучших в мире. Много интересной информации о постскриптовских и других компьютерных шрифтах вы можете найти в книге одного из ее авторов Ю. А. Ярмолы «Компьютерные шрифты», СПб, BHV–Санкт-Петербург, 1994, 204 с.

Текст, завершающий статью, программируется так:

```
%!
/m{moveto}def  /rl{rlineto}def  /rm{rmoveto}def
/hrl{0 rl}def /vrl{0 exch rl}def
/GS{gsave}def  /GR{grestore}def  /RGB{setrgbcolor}def
/S{stroke}def  /LW{setlinewidth}  def
  % Как обычно, полезные определения

/Cyrillic-Narrow findfont % нашли шрифт
60 scalefont  % пересчитали его в кегль 60
setfont       % и установили
100 500 m     % установили текущую точку
(СПАСИБО ЗА ВНИМАНИЕ)
   % положили в стек строку текста

   % начали делать красивую рамку
GS
  -10 -10 rm           % оступили влево и вниз
  dup                  % скопировали строку
  stringwidth pop      % измерили ее
    % слово stringwidth дает нам два параметра,
    % один из которых сейчас не нужен, и его
    % приходится убирать
  20 add dup           % это ширина рамки
  hrl 64 vrl neg hrl   % провели всю
  closepath            % рамку
  GS 0.7 0.8 0.9 RGB fill GR
     % фон светло-серо-синий
  0.6 0.6 0.8 RGB 3 LW S
     % сама рамка немного темнее
GR
0.5 LW dup     % снова скопировали строку
GS 1 0.4 0 RGB dup show GR
   % текст напечатали красным с небольшим
   % оттенком желтого
0.8 0 0 RGB false charpath S
   % обвели буквы более темным красным
showpage
```

Сейчас в Постскрипт переведено большинство шрифтов, накопленных мировой полиграфической культурой. Правда, началась «новая эра», - появилась возможность непропорционального изменения шрифтов (в той схеме, которой мы пользуемся сейчас, мелкие размеры имеют слишком светлые буквы, а крупные размеры - слишком жирные).

Интерпретатор GhostScript/GSView

Как уже говорилось, существует легко доступный через Интернет интерпретатор Постскрипта, разработанный фирмой Aladdin Enterprises, и его оболочки, разработан

ные для X11 Т. Тейзеном и для MS-Windows и OS/2 Р. Лэнгом. Эти программные средства доступны по адресу

http://www.cs.wisc.edu/~ghost/

Размер инсталляционного файла gsview.exe составляет около 3.5 М.

Заключение

Я рекомендую вам попробовать программирование на Постскрипте. Это расширит ваш программистский кругозор, даст вам возможность легкого изготовления иллюстраций, оформительских элементов, орнаментов и т. п.

Рисунок 4.

Для меня самого очень важно, что иногда вычислительная программа может готовить картинки, показывающие результаты расчетов, прямо в виде программы на ПостСкрипте, и они немедленно оказываются готовы для просмотра на экране и печати бумажной копии (Рисунок 4).