适用于IBMPC及其兼容机

# 电子排版及印刷技术

# Post Script的使用与实现

郭广龙

北京希望电脑公司

适用于IBMPC及其兼容机

# 电子排版及印刷技术
# Post Script的使用与实现

郭广龙

北京希望电脑公司
一九九一年八月

★北京市新闻出版局
准印证号：3061-90061

★订购单位：北京8721信箱资料部
★邮　　码：100080
★电　　话：2562329
★传　　真：01—2561057
★乘　　车：320、302、332、路车
到海淀黄庄下车
★办公地点：希望公司大楼一楼往里走
101房间

# PostScript语言指导及范例

## ——译者前言

当前是一个信息的时代，其标志就是信息的快速准确传递，而报纸、书籍、杂志等印刷出版物作为重要的传播媒介，也被迫朝着快速准确的方向迈进。于是，一种主要的实现手段——电子排版及印刷，正在受到越来越广泛的注目，一批排版系统也应运而生，但如何衡量一个排版系统的质量高低呢？人们急切地盼望着一个描述详实而又准确的标准。

令人欣慰的是，这种标准终于出现了，它就是PostScript——一种与设备无关的表示印刷页的页面描述语言，而本书则提供有助于你理解PostScript语言不可多得的说明性材料。

本书实际上是两部文献：PostScript语言指导和PostScript语言范例。

《PostScript语言指导》是对PostScript语言及其图形原语的简单的、非正式性的介绍。它的表达风格及层次主要是面向希望能设计和实现诸如字处理软件包、图形图象、CAD/CAM绘图系统等应用的程序设计人员，在阅读本书时，如有条件，最好能在PostScript输出设备（如Apple LaserWriter）上实践其中的例子或其变种程序。

《PostScript语言范例》，正如其名，是一本展示PostScript用法的程序实例集。这些所选实例既用来展示PostScript的作用范围，又可成为你设计的应用程序包中有用的组成部分。书中的实例提供了诸如描绘高质图形、高效排字技巧，维护真正的设备无关等技术手段。在阅读本书时，如有条件，最好也能在PostScript输出设备上实践其中的例子或某变种程序。

PostScript既是页面描述语言，同时也是一种程序设计语言，在解释程序的作用下，把想象中的任何所希望的页格式和内容传送到输出设备上印出。

传统的印刷艺术是丰富的，其技术也历经几个世纪的演变和进化，而Adobe System公司的PostScript则为电子时代的印刷提供了工具。毫无疑问，本书的问世，不但能大大提高你开发令人兴奋的技术的能力，而且有助你享受探索电子印刷世界的进程。同时，其间暗示的多彩思想为丰富你的思路及创新打下了基础。

编译者

# 目　录

## 第一篇　POSTSCRIPT语言指导

## 第二篇 PostScript语言范例

# 第一篇　PostScript语言指导

## 第一章　简　　介

作为一种程序设计语言，PostScript能够把任何所需页的描述传送到打印设备。它既拥有能以任意方式结合的大量的图形操作符，同时也包含变量并允许用它们构造更复杂的过程和函数。

PostScript页描述是在解释程序作用下运行的程序。PostScript程序通常可由在其它计算机上运行的应用程序产生。然而，许多PostScript打印设备，包括 Apple LaserWrite，都有一个可使用户直接用PostScript编程的交互状态。

### 1.1　作为页面描述语言的PostScript

PostScript含有大量的图形操作符，以便使之能精确地描述所需页形式。这些操作符可控制三类图形的放置：

- 正文　各种字型的正文可以任意大小和方向放在一页的任何位置。
- 几何图形　PostScript 图形操作符可构造几何图形。这些图形操作符能够描述任意大小、方向和宽度的直线、曲线，也能够描述任意大小，形状和颜色的填充空间。
- 样本图象　任意大小和方向的数字化相片、手写图案及其它图象可放在一页的任何位置。

所有的图形都能够很容易地旋转，收缩及裁剪到输出页的指定部位。

#### 1.1.1 PostScript成象模型 (Imaging Model)

成象模型就是同图形系统的设计相结合的规则集合。PostScript成象模型非常近似于我们手工绘图时本能采用的模型。

PostScript模型认为一幅图象是用墨水在一页的指定区域绘制产生的。这些墨水能用来写字，画线、涂充图形或者表示半调色的相片。墨水本身可以是黑的、白的、彩色的或某种灰度的。在放置在页上之前，这些元素还可以用任一形状的边界所修剪。一旦以所需格式生成页，就可以把它在输出设备上印出。

对于PostScript成象模型的实现，存在着三个重要的概念：

1．当前页 (Current Page)：当前页是PostScript绘图的“理想页”。它独立于所用打印设备的输出能力。

程序刚开始执行时，当前页是全空的页。PostScript绘图操作符(Painting operator)在当前页上放置一些标记，这些标记可以隐盖它们所覆盖的标记（mark），一旦绘完当前页，即可送至打印设备，产生所期待的页效果。

请记住无论标记是什么颜色的——白的、灰的、黑的或彩色的——它都是以不透明的方式放在当前页上的。

2. 当前路径(Current Path):当前路径是共同刻划形状及位置的相连的或不连的点、直线和曲线构成的集合。当前路径定义的图形不受任何约束：它们可凸可凹，甚至自交。当前路径中的元素是以它们在当前页中的位置来说明的，所用打印设备的分辨率不会约束当前路径的定义。

当前路径本身并非是当前页的一个标记。PostScript路径操作符(Path operafor)只定义当前路径，并不标记页。一旦定义了路径，即可绘制到当前页（沿路径画线），或被填充（产生墨水填满的实心域），或者用作裁剪边界。

3. 裁剪路径：当前裁剪路径是可绘区域的边界。最初的裁剪路径与打印设备的缺省页尺寸相匹配，之后可变为任意所需大小和形状。如果图形操作符标记了当前裁剪路径之外的当前页，则只有落在裁剪路径之内的标记部分才能真正画到当前页上。

### 1.1.2 座标系 (Coordinate Systems)

页上的位置是用加在页上座标系的x，y对来表示的。

每一输出设备都有其刻划页上点位置的内部座标系（built-in Coordinate System），我们称之为设备空间（device Space）。设备空间依打印设备的不同而变化，座标原点或纵横坐标的刻度也没有统一形式。

PostScript当前页上的位置是用用户坐标系（user coordinate system）或用户空间（user space）来刻划的。这种座标系独立于打印设备的设备空间，在打印当前页之前，PostScript程序中的用户座标能自动地转换成打印设备的设备座标。于是，用户空间就提供了这样一种座标系，使得在描述其间的页时，可以不必考虑页输出所用的特定打印设备。

PostScript用户空间可以有如下三种变化：座标原点可移到用户空间中的任意一点；座标轴可向任何方向旋转；座标轴可用任意尺寸来刻度，甚至于x、y方向上的刻度尺寸不同。高级用户还可以规定从用户空间到设备空间的任意线性变换。因此，PostScript程序中的座标依当前页而变化；其间的座标系可以移动，旋转，放大或缩小。

## 1.2 作为程序设计语言的PostScript

大约三分之一的PostScript语言是用于图形部分的，其余部分则构成了一个完全通用的计算机程序设计语言。PostScript语言包含有许多其它程序设计语言的成份，但同FORTH语言最相似。

### 1.1.1 POSTSCRIPT栈 (Stack)

PostScript保留了一块称之为“栈”的内存，以保存其正在使用的数据。栈操作时犹如一叠书，最后放进书叠的书最早取出。与此相似，放入栈中的数、字串或其它数据也是以相反次序取走的，即最后进栈的数据最早被检索。

1. 后缀表示法 (Postfix Notation)

作用于数或其它数据的PostScript操作符，如add和sub，需从栈中检索数据。要使用操作符，必须先将所需数据（即操作数operands）放进栈，然后调用操作符，操作符也把其运算结果放在栈里。这种操作数必须在操作符之前说明的程序设计方法，叫做后缀表示法。

2. POSTSCRIPT数据类型 (Data Type)

PostScript支持多种常见于其它语言的数据类型，如实数，布尔数、数组和字符串。此外 PostScript还可定义目标类型，如字典（dictionary）和标记（mark）。欲了解所有

PostScript数据类型及实体类型的描述，请参考《PostScript语言参考手册》一书。

POSTSCRIPT的灵活性 (Flexibility)

3. PostScript是一种非常灵活的语言。那些虽不存在，但却对应用程序很有用的函数，可以象其他PostScript操作符一样地定义和使用。因此，PostScript并不是一个把应用程序限制在其界线内的固定工具，而是一个能随任务而变化的环境。页描述的各个部分都可用于组成更复杂的页。这些部分既可以原有形式使用，也可移动、旋转、放缩以构成新成份页的金字塔。

可打印程序 (Printable Programs)

4. POSTSCRIPT程序完全是用可打印的ASCII字符写的。这使之能被大多数通讯系统和计算机文件系统以通常的正文文件形式来处理。除此之外，这也保证PostScript程序能很容易地以程序结构方式阅读。

# 第二章 栈与运算

如同所有的程序设计语言，PostScript程序设计语言也作用于各种类型的数据，如数、数组，字符串和字符。PostScript作用的这类数据又叫PostScript实体(PostScript object)。

## 2.1 POSTSCRIPT栈

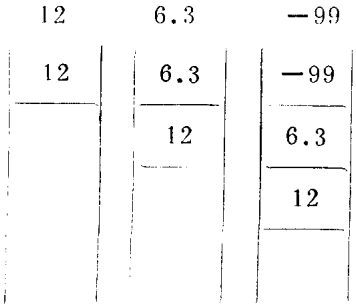
栈是为即将被POSTSCRIPT使用的数据设立的一块内存区域。该区域中的项以后进先出方式组织。这种类型的数据结构即为后进先出的LIFO栈。

LIFO栈如同一叠书。当书被叠加时——先是马克·吐温的，然后是狄更斯的，然后是海明威的，——只有最后叠加上的放在最上面的书才是真正可取的书。

加数入栈

任何在POSTSCRIPT源文件（即包含POSTSCRIPT程序的正文文件）中出现的数都是放在栈中的。例如，如果一个源文件包含下述行：

```
12  6.3  -99
```




POSTSCRIPT栈

当解释程序自左向右读进该行时将执行下述动作（见左边图示）：

1. 把数12压进栈
2. 把数6.3放进栈，同时把12压下一个位置。
3. 把数－99放进栈，同时把头两个数压下一个位置。

数－99现在是在栈顶，以备使用，其他数虽也在栈中，但只能以适当的次序取出。在使用栈时请记住任何类型的POSTSCRIPT实体都能放在栈中，包括数组、字符串以及更新奇的POSTSCRIPT实体，如字典。为了方便讨论，在本指导手册的第一、

二章中，我们主要集中于对数的操作。

记住，空格、制表符和换行符被用作POSTSCRIPT实体的分界符。其他字符，如小括弧、中括弧，在某些环境下也可作为分界符，在学习本指导手册的过程中我们会讨论到这些。

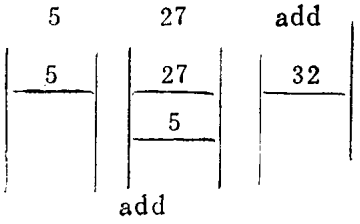
## 2.2 运算

POSTSCRIPT操作符是能引起POSTSCRIPT解释程序执行某种动作的字，如同其他语言中的命令或过程。当解释程序读进源文件中的字时，便在内部字典中查找该字；判别它是否是个操作符名。如该字出现在字典里，解释程序就执行与该名相关联的指令，然后再读进源文件中的下一个字。有关POSTSCRIPT字典的详细内容请参阅第四章。

1. add和sub

POSTSCRIPT操作符是在栈中寻找所需数，即操作数的。通常情况下，操作符都是先从栈中移走它的操作数，再把运算结果放回栈中。

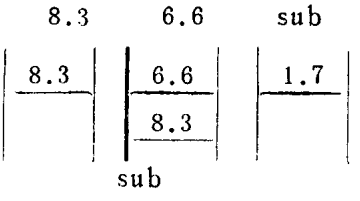
例如，操作符add使得POSTSCRIPT先从栈中移走头两个数，相加，再把其和放回栈。因此，下面的程序行将如下图影响栈。




```
5  27  add
```

先是5然后是27压入栈，尔后操作符add用它们的和替代了它们。

POSTSCRIPT操作符也是以类似方式工作的下面的程序行将如左图影响栈。




```
8.3  6.6  sub
```

先把数8.3和6.6压入栈，尔后操作符sub用下面的数减去上面的数，把其压回栈。

2. 栈表示法 (Stack Notafion)

PostScrip栈内容一般是用一行数（或其他数据）表示的，栈顶位于右部。因此，一个6为顶，143.9在其下，−800在更下的栈打印出来如下：

```
−800  143.9  6
```

注意，这种次序排列的数与它们最初进栈时的次序一致。

同样，栈上操作符的作用效果可以这样表示：先是栈的初始状态（运算执行前），然后是操作符名，最后是操作符执行后的栈内容，采用这种方法，add的作用效果可表示为：

```
5  27  add⇨32
```

3. 其他运算的操作符

除了add和sub，POSTSCRIPT中还有大量的运算操作符，包括：

div　用栈顶的头一个数除第二个数。如：

```
13  8  div⇨1.625
```

idiv　用栈顶的头一个数除第二个数，只保存商的整数部分。

```
2  53  div⇨8
```

mod　用栈顶的头一个数除第二个数，只保存余数。

12　10　mod⇨2

操作符mod和idiv的操作数必须是整数。

mul　把栈顶的头两个数相乘，其积入栈

6　8　mul⇨48

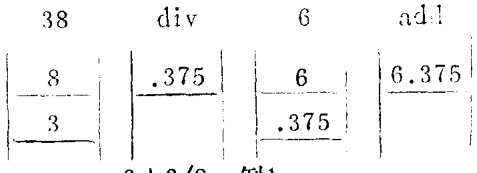
neg　把栈顶数的符号取反

−27　neg⇨27

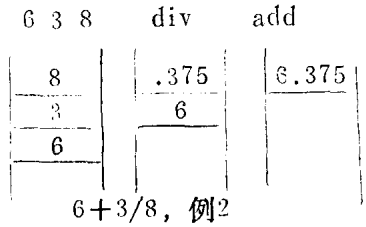
这些运算操作符在本指导手册中会经常使用。关于全部POSTSCRIPT 运算操作符，包括sqrt，exp，ceiling和sin的详细描述，请参阅《POSTSCRIPT语言参考手册》一书。

4. 更复杂的运算

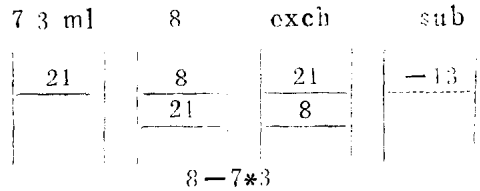
POSTSCRIPT 中栈的使用使得运算过程的执行享有一定的自由度。例如，如果想在POSTSCRIPT中计算6+(3÷8)，以下两个程序均能在栈中得到正确的答案。




6+3/8，例1




6+3/8，例2




8−7*3

·3　8　div　6　add

·6　3　8　div　add

在第一种情形下，先把3和8入栈，前者除后者，6入栈，再求它与其下的商之和。

在第二种情形中执行的是同样的操作，只是我们先把这三个数都压进栈，然后调用操作符div，使得第二个数（3）除栈顶的数（8），再把最上的两个数（6和.375）相加。

同样，方程8−(7×3) 至少也可用有两种方式表示：

·8　7　3　mul　sub

·7　3　mul　8　exch　sub

第二种方式引进了一个新操作符：exch，该操作符用于交换栈的最上面两项。注意在本例中，短语7　3　mul 先把两个数入栈，相乘后只剩下其积21在栈顶。之后数8入栈，于是栈中数的次序发生了错误。操作符sub 用第二个数减顶上的数，即21减8，得到相反的结果。而此时引进操作符exch，使栈顶的两个数次序交换，使之适合于减法中的次序。

5. 栈操作符

栈操作符系指能添加，移走或重排POSTSCRIPT栈中项的操作符。exch是我们介绍的第一个栈操作符。此外还有几种栈操作符：

clear　清除栈中所有项

6　8　12　clear⇨—

dup　复制栈顶项

6　dup⇨6　6

pop　弹出栈顶项

17　8　pop⇨17

roll　滚动栈内容。从栈中取出两个数，头一个数告诉POSTSCRIPT旋转栈的次数和

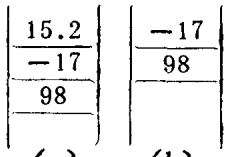
方向，第二个数说明共有几个项需要旋转。

```
7  8  9  3  1  roll⇨9  7  8
7  8  9  3  -1  roll⇨8  9  7
```

这些以及其他一些栈操作符将会在本手册中被用到。关于所有这类操作符的完整描述，请参阅《POSTSCRIPT语言参考手册》的相应章节。

## 2.3 交互式栈操作符

多数POSTSCRIPT程序都是由诸如字处理一类的应用程序所产生。然而，许多 POSTSCRIPT打印设备都有一个交互模式，允许用户直接与POSTSCRIPT解释程序对话。对这些交互式环境，POSTSCRIPT提供了可直接检查栈的操作符。




(a)　(b)

==

操作符==从栈中移出最顶项并通过通讯管道返回它（管道通常与终端相连）。因此，如果栈最终如左边的图a，当在键盘上打入==后，15.3将显示在终端上，而栈内容将变为图(b)所示。

操作符==总是尽可能地打印出栈顶项。多数实体，如数、字符串，数组等都可直接打印出。对那些不能打印的项，如字典和文件，则用其实体类型来标志。因此，如果栈顶项为一字典（以后会详细讨论字典dictionary），操作符==将在终端上打印出

```
——dictionary——
```

Pstack

第二个有用的交互式栈操作符为 Pstack。该操作符将打印出整个栈的内容。与==不同之处在于Pstack并不移走栈中的任何内容。

因此，如果栈最初如下形式：

```
6  12  -97.2  100
```

操作符Pastack将如下显示栈内容，并保持栈内容不变：

```
100
-97.2
12
6
```

Pstack和==都是多形操作符（polymorphicop erator），这样称呼它们是因为它们均能以不同类型的实体作为操作数。

## 2.4 新操作符摘要

本章及以后各章均将以该章所介绍的操作符一览表作为结束。这些览表是以新操作符的函数类型及下列信息来分组排列的。

- 操作符名
- 操作前栈内容
- 操作后栈内容
- 操作描述

两个栈内容列表是以双箭头（⇨）来分隔的栈中所用符号用以表示下列实体类型：

| | |
|---|---|
| n i j x y | 数 |
| ary | 数组 |
| bool | 布尔值 |
| dict | 字典 |
| fdict | 字型字典 |
| nam | 名字 |
| ob | 任意POSTSCRIPT实体 |
| proc | 过程 |
| str | 字符串 |

其他所用符号将另行解释。如果栈中实体可能有不止一次类型，则用斜线(/) 来分隔。因此，ary/str表示实体可能为一数组，也可为一字符串。

## 2.5 操作符一览表

栈操作符

clear　$ob_1 \cdots ob_i \Rightarrow —$

清除栈中所有项

dup　$ob \Rightarrow ob\ ob$

复制栈顶项

exch　$ob_1\ ob_2 \Rightarrow ob_2\ ob_1$

交换栈最上面两项

pop　$ob_1\ ob_2 \Rightarrow ob_1$

弹出栈顶项

roll　$ob_{n-1} \cdots ob_0\ n\ j \Rightarrow ob_{(j-1) \bmod n} \cdots ob_0\ ob_{n-1} \cdots ob_{j \bmod n}$

j次旋转n个栈顶

算术操作符

add　$n_1\ n_2 \Rightarrow n_1 + n_2$

两数相加

div　$n_1\ n_2 \Rightarrow n_1 \div n_2$

两数相除

idiv　$n_1\ n_2 \Rightarrow \mathrm{int}(n_1 \div n_2)$

商求整

mod　$n_1\ n_2 \Rightarrow (n_1\ \mathrm{MOD}\ n_2)$

求余

mul　$n_1\ n_2 \Rightarrow n_1 \times n_2$

两数相乘

sub　$n_1\ n_2 \Rightarrow n_1 - n_2$

两数相减

交互式操作符

==　　ob⇨—

破坏性地显示栈顶项

pstack　$ob_1 \cdots ob_i \Rightarrow ob_1 \cdots ob_i$

显示栈内容

# 第三章　初始绘图

POSTSCRIPT 语言是为产生图形图象而设计的。因此，在学习本手册过程中，你将发现该语言中含有大量的图形操作符。

欲用POSTSCRIPT 绘图，首先要在称之为当前页的理想绘图面上建一条路径。路径是直线和曲线的集合，这些直线和曲线或是定义一个待填充的区域，或是表示当前页上画出的一条轨迹（路径和当前页的更完整的讨论，请参阅《POSTSCRIPT语言参考手册》一书）。

建完路径，就要确定如何用它。我们可以沿当前路径画一条某种宽度的线，也可以填充该路径以产生一个实心的图形。

就这样地交替执行这两步——建路径，然后画或者填充它——直到在当前页上绘出所需的所有事情。一旦绘完当前页，便可以把它打印到一张真正的纸上了。

## 3.1　画线

让我们先从简单的任务开始：画一条 5 英寸的竖线，下面的程序完成此任务：

```
newpath
    144  72  moveto
    144  432  lineto
stroke
showpage
```

让我们逐行解释该程序。

首先调用newpath操作符，置空当前路径，开辟一条新路径。

然后，通过在当前页上移动一支虚幻的“笔”来建立该路径，该笔在当前页上留下代表当前路径的非标记性轨迹。在某一特定时刻，笔所指向的当前页上的位置叫做当前页上的当前点（current point）。

首先用moveto建路径

```
144  72  moveto
```

操作符moveto从栈中取出两个数作为只移到的x，y座标，于是，此座标变为当前点。

在POSTSCRIPT的缺省座标系里，座标原点位于当前页的左下角，x 轴向右，y 轴向上，单位长度为1/72英寸。因而，第二行程序先把数144和72推进栈，尔后移到当前点，即距离当前页在下角向右2英寸（144/72），向上1英寸（72/72）的位置。

第三行上的操作符lineto

```
144  432  lineto
```

该操作符添加一段对当前路径的操作，使当前点变为栈上的数，此例中为144，432。于是，作为该操作符参数的点成为当前点。

注意操作符lineto并非真在当前页上画，它仅仅是对当前路径添加一个线段而矣。

第4行的操作符stroke使得所建路径画到当前页上，该路径就变成了一条竖线。

最后，showpage打印当前页，即一条线。

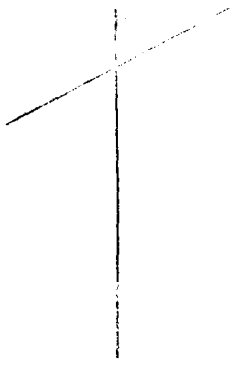
从画线过程我们可以看到以下三步：

1.用newpath，moveto和lineto建立一条POSTSCRIPT路径。

2.在当前页上stroke该路径。

3.用showpage打印当前页。

1． 两条线



如左图所示，下述程序画出两条线

```
newpath
  72  360  moveto
  144  72  rlineto
  144  432  moveto
  0  -216  rlineto
stroke
showpage
```

此程序与第一个程序相似。前两行清空当前路径，然后把当前点移到距页左下角右1英寸，上5英寸的位置。

```
newpath
72  360  moveto
```

之后一行包括一个新的操作符，rlineto

```
    144  72  rlineto
```

此操作符与第一个程序中用到的操作符lineto相似，只不过栈中的数表示的是相对于当前点的x和y位移，POSTSCRIPT中的rmoveto也与moveto相似，不同之处也仅在于栈中数表示的数是相对于当前点的位移而矣。

因此，上面的程序行在当前路径上添加一个线段，把当前点的位置向右延伸2英寸，向上延伸1英寸。

下两行程序

```
    144  432  moveto
    0  -216  rlineto
```

把当前点移到第一个线段之上而后添加第二个线段，使当前路径向下（注意参数y为负数）延伸216个单位（即216/72＝3英寸）。

此时已经有了一条由两条相交线段，在没有使用操作符stroke之前，这些线段都是不可见的。注意，当前路径并非是连续的。一条POSTSCRIPT路径并非一定仅仅是一个相连的片断，它可以由当前页上的若干线段和曲线组成。

最后，程序画出路径并打印当前页。

2． 方框

下列程序在页中央画一个1英寸的方框：

方框

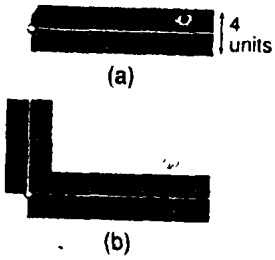
```
newpath
    270   360   moveto
     0    72    rlineto
    72    0     rlineto
     0   -72    rlineto
   -72    0     rlineto
     4    setlinewidth
stroke   showpage
```

该程序先把当前点移到页的中央附近，然后向上、右、下、左各移1英寸，形成一条方框状的路径，之后画出路径并打印页。

程序的第七行引进了新操作符：

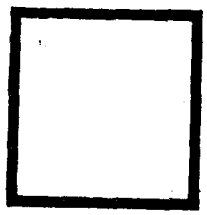
```
4   setlinewidth
```

操作符setlinewidth说明了画路径时所用的线宽，在本例中，线宽为4/72英寸。在调用新的setlinewidth之前，画在页上的所有线宽均保持不变。




你可能已注意到，这个方框有一点缺陷：左下角有一个凹口。这是因为线存在宽度。

一条4个单位宽的线段向当前路径的两边各扩展2个单位（见左图a）。当方框的第一条及最后一条线段相交时，一个不属于所画路径的任何部分的两单位方形区域将仍保留为白色（见左图b）。



A Better Box

为了避免这种问题，应使用新操作符：closepath。

```
newpath
    270     360     moveto
    0       72      rlineto
    72      0       rlineto
    0      -72      rlineto
    closepath
    4 setlinewidth
stroke showpage
```

此程序等同于前一个程序，只是通过使用closepath使方框密合。操作符closepath向当前路径添加一条路段，用以连接当前点和最后一个用操作符moveto指定的点。通过采用斜连接来密合路径，避免了第一个方框中出现的凹口。它有可能改变了POSTSCRIPT连接线段的方式。欲知POSTSCRIPT是如何做到这一点的，请参阅本指导手册的第九章。

### 3.2 填充图形

迄今为止我们已学会建路径并把它们画到页上的方法，然而，POSTSCRIPT路径还可被填充，下面的程序除了一行之外均同上一个相同：