G53DOC
Lab Exercise 1:Postscript

Steven R. Bagley
(based on material by David F. Brailsford)

Introduction

This document contains three simple PostScript exercises for you to try. These are non-assesed and are purely to give you experience in writing PostScript programs. Assistance will be available in the lab to help you if you get stuck. In addition, the files `usefulps.pdf`, `exps.pdf` and `exps2001.pdf` contain information that will be useful in getting started. All exercises make use of sample files that can be found in the supporting tarball.

PostScript programs can be editted with a standard text editor and viewed using a variety of tools. The easiest method on the school machines is to use the freely available **GhostScript** interpreter which is installed on the school machines and is available for download for Windows and Linux. Under MacOS X, Preview will happily open a PostScript file and display it on screen.

1. Text and Fonts

In this example, we make use of PostScript's font and text handling to create a diagram linking type designers to their fonts. (Note for this to display correctly it will need to be run on the School's Solaris servers.)

Most of the code has been provided but you will need to write a procedure body for **boxtext** in the framework set out within the file `boxtext.ps`. (Make sure you read **very carefully** all of the comments in that file. The end result should be as shown in the first appended sheet at the end of this coursework (em at least, under the Solaris servers.)

2. Pie Charts

The Acme ice-cream company wants to display a pie chart of its percentage sales across the four flavours that it makes. To impress the shareholders it is decided to colour the pie segments in accordance with the ice cream flavours. A Web programmer points out to the PostScript team the existence of `www.htmlhelp.com/cgi-bin/color.cgi` which gives required RGB values for various RGB colours, in the form of 24-bit integers expressed in hexadecimal. Each of the RGB hues is represented from left to right by pairs of hexadecimal digits. Thus, each hex pair represents an integer number of 256ths for the intensity of Red, Green and Blue respectively. After much argument it is decided that the company's standard flavours of Strawberry, Vanilla, Toffee and Lemon will be represented by the Web shades of Deep Pink, Cornsilk, Peru and Lemon respectively.

The template for the piechart PostScript program is in `rawpie.ps`. You need to develop the `pchart` procedure step by step until the calls at the bottom of the file give the sort of effect that is appended to this coursework. Credit will be given for good placement of text which must be kept horizontal, as shown. Make the text placement be dependent on the angle of the slice and the radius of the pie in some way. Don't just kludge things so that it works OK for the 2 and 3 inch radius piecharts that you are given but would fail miserably for other radii.

Hint: Don't forget that trig. functions sin and cos are available to you for helping to place the text. Also the conversion of hex values for the Web RGB shades into the fractional values needed by PostScript will be greatly helped by use of the bitwise logical operators i.e. `and`, `or` and `not`, and also by the `bitshift` operation.

3. Drawing Stacks

PostScript has the advantage that, being a graphics language, it could, in principle, display the state of the argument stack whenever a procedure is entered. For instance, it would be helpful as a debugging aid for the `shadow` routine in `exps.pdf` to be able to display arguments and values as shown below:

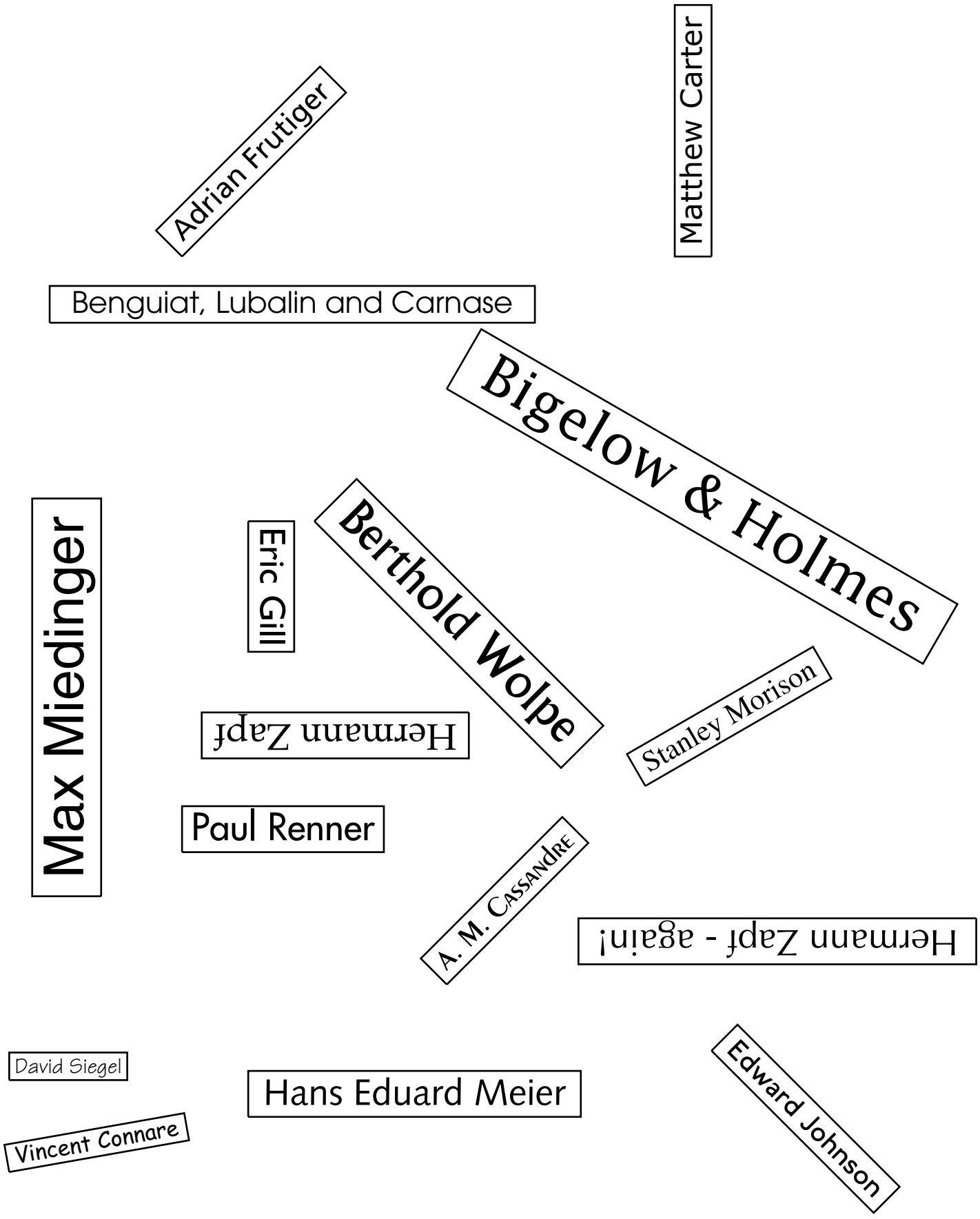
xc 320
yc 400
startgr 0.95
endgr 0
incgr .05
p 20
str CPU
ft /AvantGarde-Book

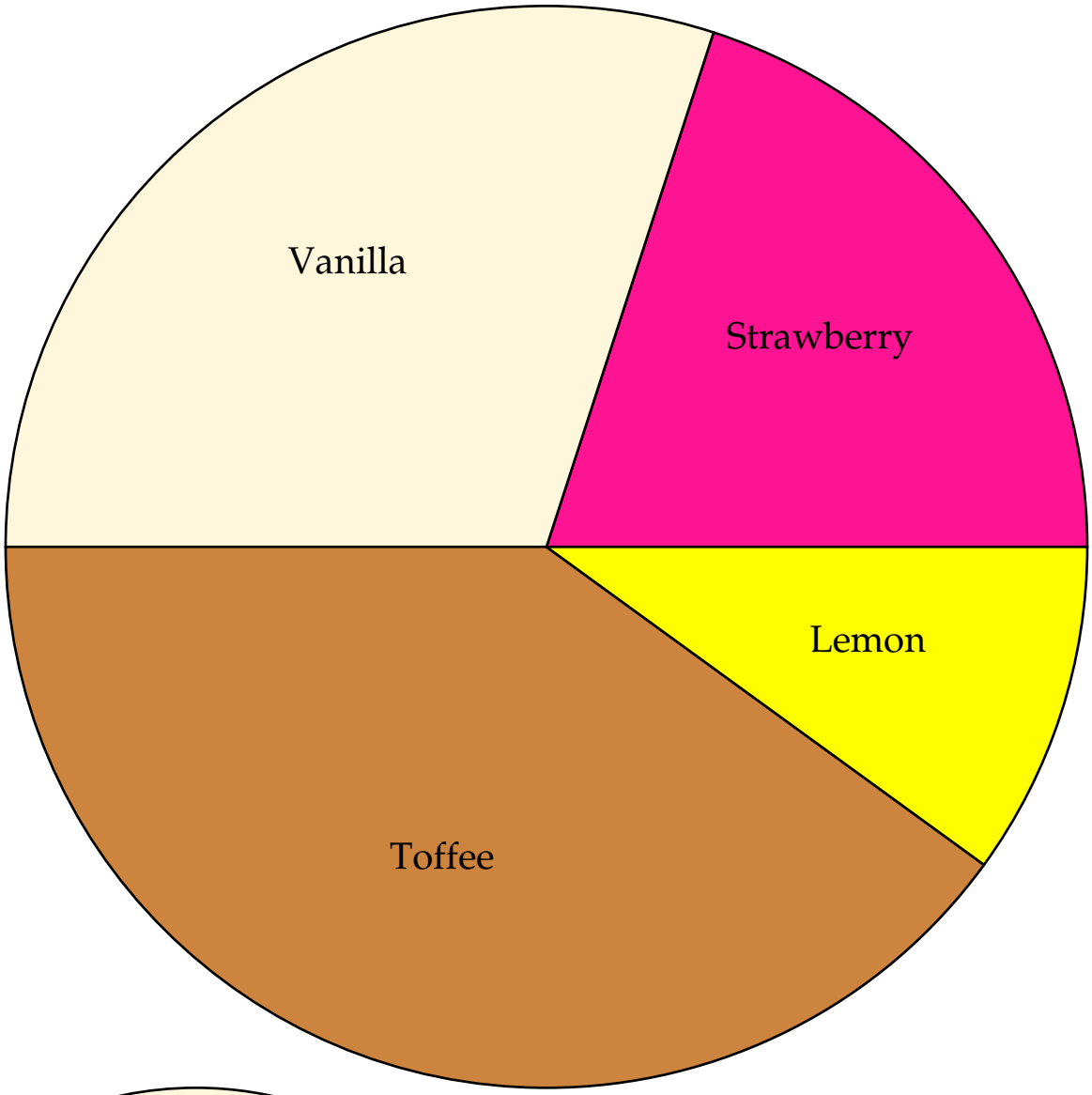
As a first step towards implementing such a facility you are asked to implement a PostScript function called `drawStack`. Determining the type of input arguments to functions and converting their dictionary names and their values to strings can be just a little tricky so `drawstack` is a simplified starting point. It will take as arguments a `null` terminated list of arrays, of arbitrary length, and each member of this list will be an array of two strings representing a formal parameter name and its value. The subsequent four parameters are the point size (your function should scale up nicely to any point size), the leading (which in this case we shall define as the difference between the point size and the height of each cell in the stack diagram), and finally the x and y positions of the centre point of the base of the diagram.

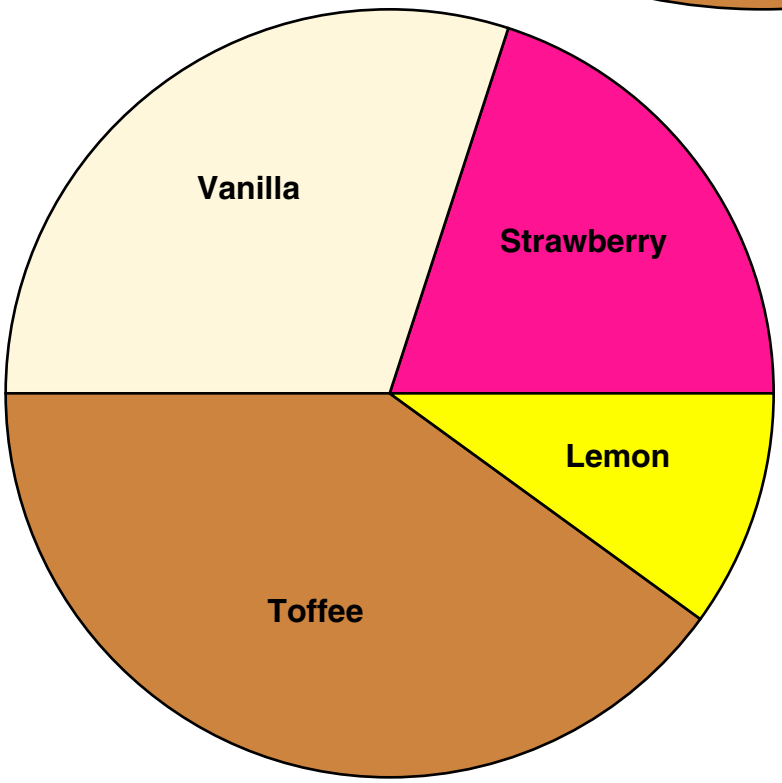
Given the skeleton of a PostScript program in `stack.ps`, complete the function so that it draws stacks on screen like the one shown above. Note that you are already supplied with the function `choosefont` to allow the bold and upright fonts in your boxes to be changed at any stage.

Requirements and Hints

- The text should all be centred within the boxes, with the boxes being slightly wider than the widest line of text.
- The argument name must be in bold and the argument value in the upright version of the chosen font with three spaces between name and value.
- Note that PostScript draws characters from the bottom left of the baseline…
- `leading` + `pointsize` is the required value for the height of the boxes.
- Put the values for poinsize, leading etc into dictionary variables (just as in `shadow`) the moment you enter `drawStack` but leave the set of stringpair arrays on the stack (you don't know in advance how long this will be!) and pull each pair off the stack one at a time for processing.
- Think about the order you draw things in…


Adrian Frutiger
Matthew Carter
Benguiat, Lubalin and Carnase
Bigelow & Holmes
Max Miedinger
Eric Gill
Berthold Wolpe
Hermann Zapf
Stanley Morison
Paul Renner
A. M. Cassandre
Hermann Zapf - again!
David Siegel
Hans Eduard Meier
Edward Johnson
Vincent Connare


Vanilla
Strawberry
Lemon
Toffee


Vanilla
Strawberry
Lemon
Toffee