$Id: asg3-draw-inherit.mm,v 1.60 2010-02-05 13:49:49-08 - - $
/afs/.cats.ucsc.edu/courses/cmps109-wm/Assignments/asg3-draw-inherit

1. Overview

In this assignment you will implement a graphics package which reads in commands to store various object specifications in an object table. Using inheritance, you will have each of these objects respond to a draw command which will output Postscript code. You will read specifications from a files, and for each input file, generate a Postscript file.

2. Program Specification

We present the program specification in the form of a Unix **man**(1) page.

NAME

draw — drawing program producing Postscript pictures

SYNOPSIS

draw [**-@** *flags*] [*filename* ...]

DESCRIPTION

Drawing commands are read from the standard input or files and output is generated in the form of pictures in the Postscript language.

OPERANDS

Operands are filenames. Each file thus specified is read in sequence and an output file is generated in the current directory. The default suffix of **.dr** is removed, but no complaint is made about other suffices. The output suffix generated is **.ps**.

OPTIONS

The **-@** option is followed by a sequence of flags to enable debugging output, which is written to the standard error. The option flags are only meaningful to the programmer.

COMMANDS

All output is generated for 8.5 × 11 inch letter paper with a printable area of 8 × 10.5 inches centered on the paper. All coordinates are relative to a (0,0) point 1/4 inch above and to the right of the lower left corner of the page. The x-axis increases to the right and y-axis increases upward. So the upper right end of the printable area is at (576,756) points. In the syntax specifications below, literal commands are represented in **Courier Bold** font and variables to be substituted are written in *Italic* font. When a symbol is enclosed in brackets, it is optional.

Sizes, heights, widths, lengths, and diameters are all numbers, as are the polygon and drawing coordinates. Font size and line thickness are measured in points, the drawing angle is measured in degrees, and all other numbers are measured in inches. An inch is 72 points. The thickness mentioned is the thickness of the line used to draw the object, defaulting to 2 points.

The **define** command creates an object and stores it in a symbol table.

# ...

The character hash (#) introduces a comment. All characters from the hash to end of line are ignored. A comment may appear on a line by itself or after a command. Empty lines and lines consisting solely of white space are ignored.

define *name* **text** [*size*] *font words...*

A text object is created by concatenating all of the words together into a single string separated by words. It is associated with the given Postscript font and size. A font is always a name, never a number. If no font size is given, a default of 12 points is used. The coordinate on the draw command is always the left end of the text. Valid font names will vary from one printer to another. Expect **ghostview** to crash if given an invalid font name.

define *name* **ellipse** *height width* [*thick*]

An elipse is created with the give height diameter and width diameter. The draw command's coordinate is always the center.

`define` *name* **`circle`** *diameter* [*thick*]
A circle with the given diameter is created. It is just an ellipse with identical height and width.

`define` *name* **`polygon`** *x1 y1 x2 y2 x3 y3 ...* [*thick*]
A polygon is created, with the first point always implied as (0,0). All other coordinates are relative distances from the previous coordinate, and the number of coordinates must be even. If an odd number of values are given, the last one is taken to be the line thickness.

`define` *name* **`rectangle`** *height width* [*thick*]
A rectangle is created with the lower left corner assumed to be the (0,0) point. It is just a polygon with a list of three coordinates, not counting the lower left position.

`define` *name* **`square`** *width* [*thick*]
A square is just a rectangle with all sides equal. It is just a rectangle with identical height and width.

`define` *name* **`line`** *length* [*thick*]
A horizontal line is drawn rightward with the given length. It is a horizontal polygon with one point other than (0,0).

`draw` *name x0 y0* [*angle*]
The given object is drawn from the given coordinates, with the first point of a polygon or the center of an ellipse. If an angle is given it is rotated by the specified number of degrees.

`newpage`
The rest of the drawing goes on a new page.

EXIT STATUS

0 No errors were detected.

1 Error messages were printed to the stderr.

3. A Tour of the Code and Data

You have been provided with the following files :

`code/Makefile`
This is familiar and unchanged, except for the names of the files to be compiled. Always submit using the **`submit`** target so as not to forget some files. Frequently use the **`ci`** target which checks in your files and also runs **`checksource.`**

`code/main.cc`
The main function scans options and then iterates over each file in turn. Note that in this assignment, you either process **`cin`** to **`cout`**, or you iterate over all files mentioned in **`argv`**, processing **`.dr`** input files to **`.ps`** output files. As files are parsed, each line of input is interpreted. Note that the **`TRACE`** facility should be used frequently, as should **`dbx`**(1).

`code/trace.{h,cc}`
The trace facility is familiar to you from previous projects.

`code/util.{h,cc}`
The utility module is also familiar. Note that in this assignment you are ***not*** using **`vector`**s any more. Instead, you will use the datatype **`list`**, which is amenable to chomping from the front end.

`code/numbers.{h,cc}`
This contains several classes you can use to represent **`double`**s. It is essential that you keep track of dimensions properly. You have three of them : Inches, which are used in most input specifications, degrees, which are used for angular specifications, and points, which are used by Postscript for measurement. One inch is 72 points. It also contains the type **`xycoords`**, which is convenient for pairing an (x,y) coordinate, and a function **`todouble`** which converts a string to a

double or throws an exception.

code/interp.{h,cc}

The interpreter takes a command and processes it, throwing a runtime error if it recognizes that something goes wrong. It takes care of writing the Postscript prolog, epilog, and page boundaries. The interpretation function looks up a function based on a command and calls it. The functions **do_define**, **do_draw**, and **do_newpage** take care of accessing the factory to make new objects and telling those objects when to draw themselves.

There are three maps in the module: The **objectmap** maps names of defined objects onto the shapes themselves, the **interpmap** selects functions to carry out the various commands, and the **factorymap** contains pointers to the various factory functions. The function **make_object** determines what kind of object needs to be created, then calls an appropriate factory function.

You need to complete each of the other factory functions, which currently only create objects with dummy arguments. For each of these functions, scan the parameter list for validity and throw a syntax error if the number of parameters is incorrect, or if a non-number is given for any argument other than the font name and words of a text. Complete a base class before trying a derived class. Thus: **make_ellipse** before **make_circle**; **make_polygon** before **make_rectangle** and **make_line**, etc. You should try to leverage code by factoring out repeated code sequences into functions. You may add as many private member functions as is appropriate.

code/object.{h,cc}

An inheritance hierarchy is specified for the objects to be manipulated. Each of them has a ctor and a drawing function.

The constructors are completed, but the **make_list** utility functions of **rectangle** and **line** need to be completed. A rectangle has three coordinates to be put into a list. The first position is assumed to be (0,0) and is specified at the time the draw command is issued. A line is drawn horizontally to the right and has one point after the assumed (0,0).

The classes **object** and **shape** are abstract. You need to implement the drawing functions for **text**, **ellipse**, and **polygon**. The classes **circle**, **rectangle**, **square**, and **line** just inherit the drawing function from their respective base class, and so don't need to be separately implemented.

data/

Contains **draw.perl**, which you can use as a reference implementation. It also contains some output files.. The input files are in the **.score** subdirectory. Use **ghostview** to see how the Postscript files are displayed. You can also print them. To use **ghostview**, you will need to be running an X-windows client. If you are using something silly that does not have **ghostview**, you can use **ps2pdf** to convert the Postscript files to PDF for viewing.

4. Postscript

Postscript is a page formatting programming language which is, in fact, Turing complete. However you only need to know a very little bit of Postscript in order to write your program. A Postscript program consists of a prolog, some pages, and an epilog. Each page consists of a start of page, some commands, then an end of page. Each figure consists of a **gsave**, some commands to draw the picture, and a **grestore**.

Study the files generated by **draw.perl** to see what code is needed in each case. In the samples below, the left column shows sample Postscript output, and the right column describes it. `Courier` font in the left column shows boilerplate. **Courier Bold** font shows what must be substituted. In addition, echo all commands to the output file in the format they are read. A Postscript comment begins with a percent sign (`%`).

%!PS-Adobe-3.0 %%Creator: **$execname** %%CreationDate: **$currentdate** %%PageOrder: Ascend %%Orientation: Portrait %%SourceFile: **$infilename** %%EndComments	Boilerplate that appears at the beginning of file: **$execname** is the basename of the executable binary, as given by **getexecname**(3c) or **argv[0]**. **$currentdate** is the date and time the program was run, as printed out by **date**(1) with **LC_TIME=C**. **$infilename** is the name of the source file.
%%Page: **$pagenr $pagenr** 18 18 translate	Begin by labelling the current page number and translating the image to 1/4 inch from the lower left. Printers can not print all 8.5 by 11 inches of a page image.
showpage grestoreall	Appears verbatim at the end of every page.
%%Trailer %%Pages: **$pagecount** %%EOF	A file finishes with a page count so that **ghostview** can show page numbers.

gsave
   /**$fontname** findfont
   $fontsize scalefont setfont
   $xpos $ypos translate
   $angle rotate
   0 0 moveto
   (**$textdata**)
   show
grestore
	$fontname is any valid Postscript font name. See **adobe-fonts.dr** for a list of the 35 standard Adobe fonts. You can not validate a font name, since many printers have more fonts than these.
	$fontsize is specified in points.
	$xpos and **$ypos** are is the (0,0) origin relative to where the rest of the figure is drawn.
	$angle is specified in degrees.
	$textdata is the text to be printed. Parentheses (**()**) and the backslash (\) must be escaped.

gsave
   newpath
   /save matrix currentmatrix def
   $xpos $ypos translate
   $angle rotate
   $xscale $yscale scale
   0 0 **$radius** 0 360 arc
   save setmatrix
   $thick setlinewidth
   stroke
grestore
	$xpos and **$xpos** specify the center of the ellipse and **$angle** is measured in degrees.
	$xscale and **$yscale** specify how much the circle is to be squeezed into an elliptical shape.
	$radius is the radius of the circle prior to squeezing.
	$thick is the thickness of the line, in points. Given the **height** and **width** of an ellipse, the scale and radius are computed as follows:
	if (**$height** < **$width**) {
	   $xscale = 1;
	   $yscale = **$height** / **$width**;
	   $radius = **$width** / 2;
	}else {
	   $xscale = **$width** / **$height**;
	   $yscale = 1;
	   $radius = **$height** / 2;
	}

```
gsave
   newpath
   $xpos $ypos translate
   $angle rotate
   0 0 moveto
   $dx1 $dy1 rlineto
   $dx2 $dy2 rlineto
   $dx3 $dy3 rlineto
   $dx4 $dy4 rlineto
   closepath
   $thick setlinewidth
   stroke
grestore
```

`$xpos` and **`$ypos`** specify the original corner (x0,y0) relative to which all the other corners of the polygon are given (in points).
`$angle` is measured in degrees and
`$thick` in points.
Each of the rest of the corners in the polygon, (**`$dx1`**,**`$dy1`**), (**`$dx2`**,**`$dy2`**), etc., are measure in points relative the the previous point. There is one fewer of these that are corners in the polygon, since the first corner is at (0,0) relative to the drawing parameter.

5. What to Submit

As usual, submit all of your header and implementation files, **`Makefile`**, and **`README`**. Use the target **`submit`** so that you don't forget to submit some files. Use the script **`testsubmit`** to verify that your program will compile. And don't forget **`checksource`**. If you are using pair programming, follow the instructions in **`/afs/cats.ucsc.edu/courses/cmps109-wm/Syllabus/pair-programming`** about **`README`** and **`PARTNER`**.