49161 **Fall Term 1999**

Mandatory assignment

In this assignment you will make a program in SML and write a technical report documenting the program.

Form and contents of the report

You should consult the note *Documenting software* for guidelines about form and contents of the report. The report should be *short*, no more than 10 pages + program listing.

The report should be made by use of electronic text processing in order to ease the corrections and changes which will be part of the writing process.

Problem

You will make a library of SML functions for manipulation and drawing of pictures. A picture is a set of segments together with a rectangular, upright bounding box in the plane. The bounding box is not shown when drawing a picture but it is used when defining operations on pictures. We use usual rectangular, Cartesian coordinates in the plane, so points in the plane are represented by coordinates which are pairs (x, y) of real numbers. The point with coordinates `(0.0,0.0)` is called the *origin* of the plane. A picture is normally placed in the coordinate system such that the bounding box is situated in the corner of the first quadrant.

If c is a real number with $c > 0$ then a picture can be scaled by factor c by mapping each point (x, y) to the point $(c\,x, c\,y)$. The scaled picture will have width $c\,a$ and height $c\,b$ where a and b are width and height of the original picture. Scaling is used in some of the below operations in order to adjust the width or the height of a picture.

You will make the following functions on pictures:

`grid` Computes a picture directly from width and height of the bounding box and the coordinates of the end-points of the segments in the picture. For convenience of use the function should be declared such that all the numbers in the input are integers (the function must convert to real numbers as used in the value representing a picture).

`rot` Computes the picture p' obtained form the picture p by first rotating $90°$ in the positive (counter-clockwise) direction around the origin and then translating the resulting picture to the right to get its lower left-hand corner into the origin. The height of p' will be the width of p and the width of p' will be the height of p.

`flip` Computes the picture obtained from a picture by reflecting around the vertical line through the middle of the bounding box.

`beside` Computes the picture obtained from two pictures p_1 and p_2 by uniting p_1 with a copy of p_2 which has been placed to the right of p_1 and scaled to the same height.

`below` Computes the picture obtained from two pictures p_1 and p_2 by uniting p_1 with a copy of p_2 which has been placed on top of p_1 and scaled to the same width.

`row` Computes the picture obtained by placing n copies of a picture p beside each other.

`column` Computes the picture obtained by placing n copies of a picture p below each other.

`draw` Outputs a drawing of a picture to a file on the disk with specified width of the bounding box to scale the drawing and specified coordinates of the lower left corner to place the drawing on the sheet. The file should be in *PostScript* format (cf. the remarks below).

Points in the plane. Translation, rotation and reflection

The horizontal *translation* of length d maps the point with coordinates (x, y) into the point with coordinates $(x + d, y)$.

The vertical translation of length d maps the point with coordinates (x, y) into the point with coordinates $(x, y + d)$.

The rotation of 90° around the origin maps the point with coordinates (x, y) into the point with coordinates $(-y, x)$.

The *reflection* in the line $x = d$ maps the point with coordinates (x, y) into the point with coordinates $(2\,d - x, y)$.

The *multiplication* by factor c from the origin of the coordinate system maps the point with coordinates (x, y) into the point P' with coordinates $(c\,x, c\,y)$.

Output of a picture. PostScript format

The language *PostScript* is used for representing figures to be printed. The ×-shaped picture with bounding box of width 100.0 and height 150.0 containing the diagonals of the bounding box (i.e. the two segments joining opposite corners of the bounding box) may e.g. be represented by the following PostScript commands when placed with lower left corner in the point (50.0,100.0):

```
%!PS-Adobe-2.0
%%BoundingBox: 50.0 100.0 150.0 250.0

1 1 scale
newpath

50.000000 100.000000 moveto
150.000000 250.000000 lineto
stroke

50.000000 250.000000 moveto
150.000000 100.00000 lineto
stroke

showpage
```

The coordinates of lower left and upper right corners of the bounding box are placed in the `%%BoundingBox:` command. Each segment in the picture yields a `moveto` command containing the coordinates of the start point, a `lineto` command containing the coordinates of the end point, and a `stroke` command. The percent character in "`%!PS-Adobe-2.0`" must be the very first character in the file, and there must be a terminating new line character after the `showpage` command.

The conversion of real numbers to decimal form can be made using the following conversion function:

```
fun realToString x = (if Real.sign x = ~1 then "-" else "")
                     ^ Real.fmt (StringCvt.FIX NONE) (abs x)
```

When programming the function `draw` you can make an auxiliary function to build the string of PostScript commands and then use the below function `wrstring` to output the string to a picture file on the disk. (Excessive use of string concatenation (operator `^`) in building the string is inefficient, so you may first build the list of PostScript command strings for the segments in the picture and then afterwards join these strings by *one* application of the `concat` function.)

NB: Note that the `wrstring` function will overwrite an existing file "`filename`":

```
(* wrstring: string * string -> unit *)
fun wrstring(data,filename) =
   let val out = TextIO.openOut(filename) in
       TextIO.output(out,data);
       TextIO.closeOut(out)
   end
```

A picture file (in PostScript format) is shown on the screen using the `gv` ("ghostview") program (in an `xterm`):

> `gv` *filename* `&`

A picture file can also be output directly on a PostScript printer as the printer will interpret the commands and make the corresponding drawing. This will usually work on a laser printer as most laser printers are PostScript printers. The lower left corner of the sheet has coordinates (0.0,0.0) and 72 units in the coordinates correspond to one US inch on the sheet (i.e. 100 units $\approx$ 3.5 cm).

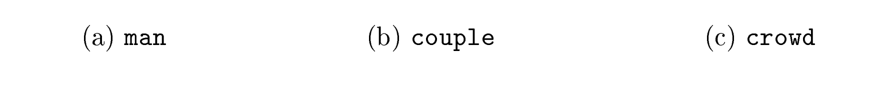
Furthermore, you may use a picture file to get the picture included into your report - provided that the text processing program supports the inclusion of PostScript graphics (the program will use the coordinates in the `%%BoundingBox:` command to determine the size and the boundary of the picture).

Examples

The functions should be used to construct and draw pictures of persons, Escher's fishes and Escher's web – as described in the following.

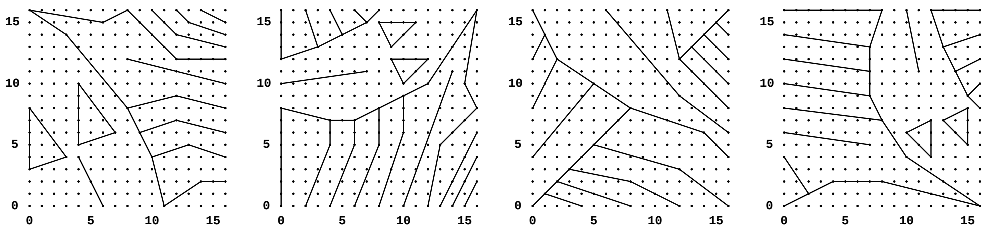
Persons

The starting point is the picture `man` shown in Figure 1. It has width 14 and heigh 20. Using the functions on pictures you should now make SML programs which draws the pictures `couple` and `crowd` shown in Figure 1.



(a) `man` (b) `couple` (c) `crowd`

Figure 1: The `man` picture and derived pictures

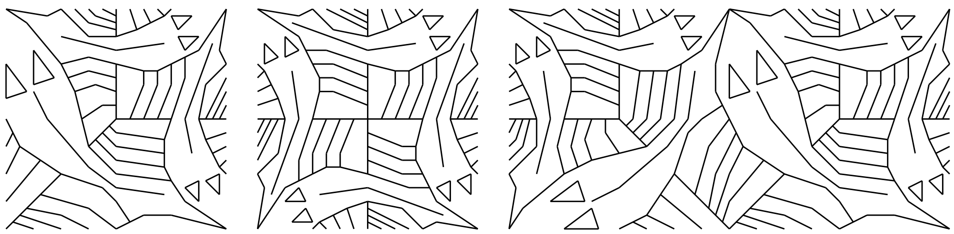


(a) Picture `p` (b) Picture `q` (c) Picture `r` (d) Picture `s`

Figure 2: Basic fish pictures

Escher's fishes

The starting point for drawing Escher's fishes is the four (16×16) pictures `p`, `q`, `r`, and `s` shown in Figure 2. By combining these four pictures we get the picture `t` in Figure 3, while the picture `a` is obtained by combining suitably rotated copies of `q`. Finally the picture `b1` is obtained by combining two suitably rotated copies of `t`.

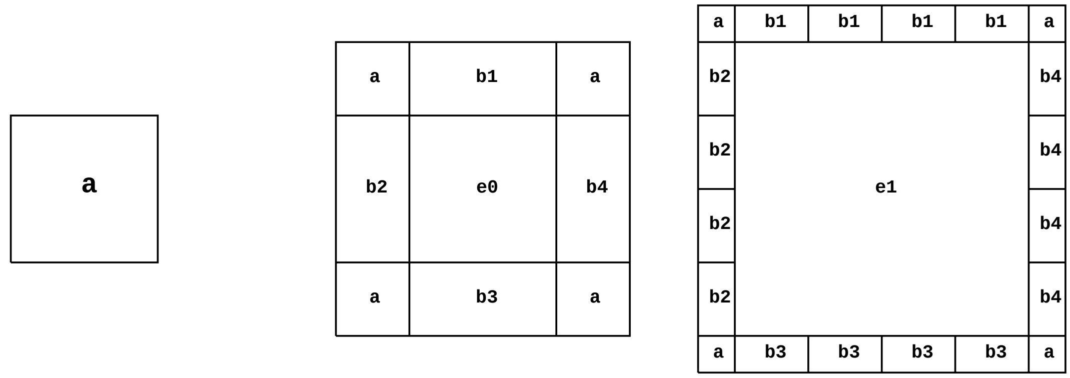


(a) Picture `t` (b) Picture `a` (c) Picture `b1`

Figure 3: Escher fish building blocks

The Escher fish pictures `e0`, `e1` and `e2` are now obtained by combining the pictures in Figure 3 as shown in Figure 4 on the next page. The pictures `b2`, `b3` and `b4` are obtained from `b1` by succesive rotations. The transition from an Escher picture to the next adds a border around the picture consisting of a picture `a` in each corner, a row of `b1`'s at the top, a column of `b2`'s at the left, a row of `b3`'s at the bottom, and a column of `b4`'s at the right. In this border there will be one `b1` on top of an `a` and two `b1`'s on top of a `b1`, one `b2` to the left of an `a` and two `b2`'s to the left of a `b2`, etc.

You should make an SML program to draw the Escher fish pictures `e0`, `e1` and `e2`.



(a) Picture `e0` (b) Picture `e1` (c) Picture `e2`

Figure 4: Building Escher fish pictures

Escher's web

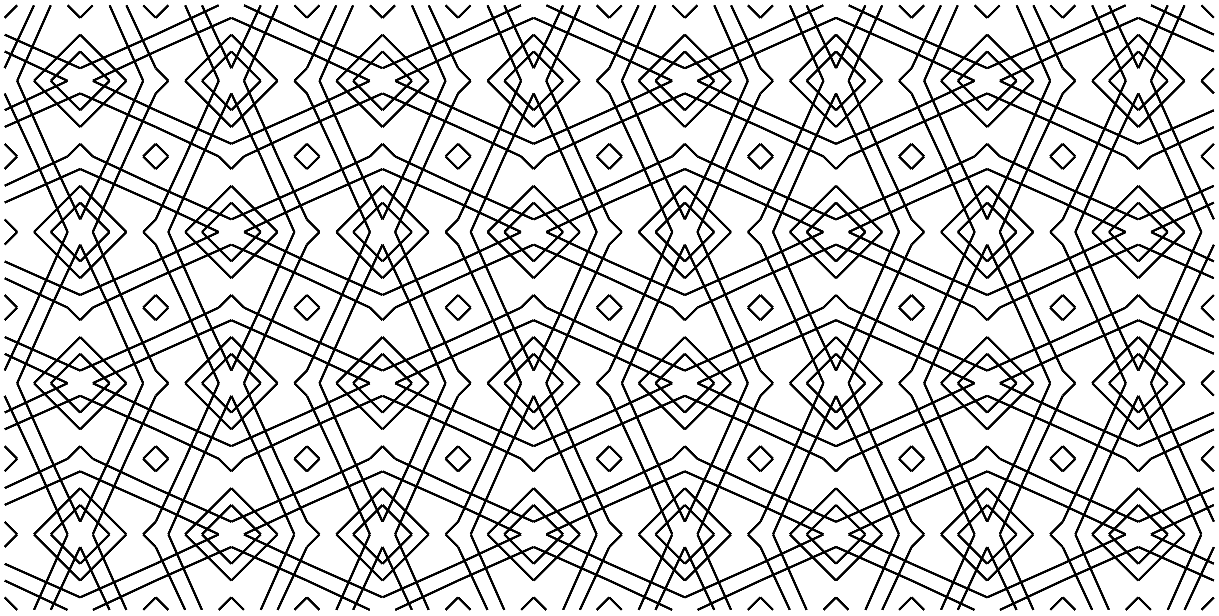


Figure 5: Escher Web

The starting point for drawing Escher's web in Figure 5 is the pattern in Figure 6. In analyzing the web you may start identifying a "big square" such that the web can be obtained by putting 8 big squares together. Next, you may identify a "small square" such that a big square can be obtained by combining 4 suitably rotated small squares. Finally, you may find out how a small square can be obtained by combining 4 tiny squares each

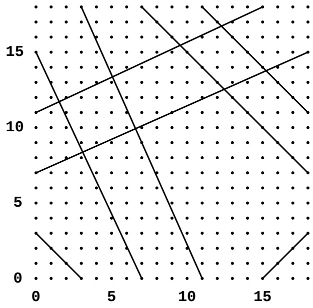


Figure 6: Escher pattern

of which can be obtained from the Escher pattern picture shown in Figure 6 by applying suitable combinations of rotations and flip'ing.