COMP 212 Spring 2015
Homework 06

1 Shapes

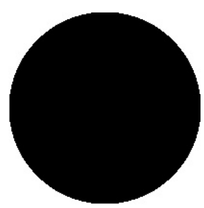
In lecture, we discussed a `datatype` for shapes which included

- a rectangle, specified by their bottom-left and top-right cartesian xy-coordinates.

- a union of two shapes, which contains all the points in both shapes

In this assignment, you will consider an extension with some additional shapes:

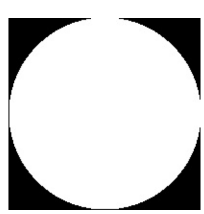
- a disc



  A disc is specified by its center point and radius; the disc specified by $((c_x, c_y), r)$ contains the points (x, y) such that

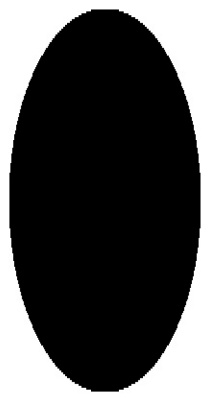
$$(x - c_x)^2 + (y - c_y)^2 < r^2$$

- a shape *without* another shape, which contains those points that are in the former but not in the latter. For example, the above square without the above dosc looks like this



- a *translation* of a shape s by (x_0, y_0), which contains all the points of s moved to the right by x_0 and up by y_0. For example, the translation of a rectangle from $(0,0)$ to $(1,1)$ by $(1,1)$ is a rectangle from $(1,1)$ to $(2,2)$.
- a *scaling down* of a shape s by (f_x, f_y), which shrinks the x direction by a factor of f_x and the y direction by a factor of f_y. For example, scaling the above disc down by $(1,2)$ gives

- a *scaling up* of a shape s by (f_x, f_y), which expands the x direction by a factor of f_x and the y direction by a factor of f_y. For example, scaling the above disc up by $(1,2)$ gives



We will represent shapes by the following datatype:

```
type point = int * int

datatype shape =
    Rect of point * point (* bottom-left and upper-right *)
  | Disc of point * int (* center and radius *)
  | Union of shape * shape
  | Without of shape * shape
  | Translate of shape * (int * int) (* x shift , y shift *)
  | ScaleDown of shape * (int * int) (* x factor, y factor *)
  | ScaleUp of shape * (int * int) (* x factor, y factor *)
```

1.1 Contains

Task 1.1 (14 pts). Extend the function

```
contains : shape -> point -> bool
```

for these new shapes. `contains s (x,y)` should evaluate to true iff the point `(x,y)` is in the shape `s` according to the above definition.

Note that, unlike in lecture, the function is now curried.

Task 1.2 (3 pts). You can test the case for `Disc` by doing

```
- writeshape(200,200, bowtie,"output.bmp");
```

to create a file `output.bmp` for the shape `bowtie` included in the support code. The first two arguments are the width and height of the image that will be printed.

You can test the cases for `Without` and `Disc` by using

```
- writeshape(415,285, example,"output.bmp");
```

to create a file `output.bmp` for the shape `example` included in the support code.

Write some additional examples that test translation and scaling, and test them using `writeshape`.

1.2 Bounding boxes

It is pretty annoying to have to calculate the width and height of a shape by hand to decide what to print. We can automate this by calculating a *bounding box* for a shape, which is a rectangle that contains all of the points of the shape.

Task 1.3 (20 pts). Define a function

```
boundingbox : shape -> point * point
```

where the result is the `(lower_left, upper_right)` corners of a rectangle such that

For all shapes `s` and points `(x,y)`, if `contains s (x,y)`
then `containsRect (boundingbox s) (x,y)`

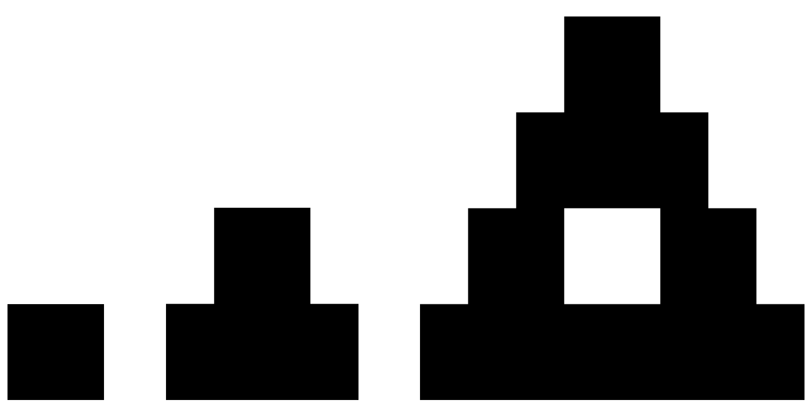
It is okay to be conservative on `Without`.

Once you have done this, you can print a shape `s` by running

```
- writeshape_bb(s,"output.bmp");
```

1.3 Fractals

The Sierpinski triangle is a fractal whose first terms are as follows:



The rule is that S_{n+1} consists of two copies of S_n along the bottom, with a third copy of S_n centered above them.

Task 1.4 (13 pts). Define a function

```
sierptri : shape -> int -> shape
```

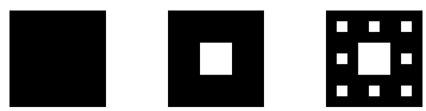
such that `sierptri s n` computes the `n`th Sierpinski triangle, starting with the shape `s` as S_0.

For example, you can test by running `sierptri sierptri_box 3`, which starts with a square like above, or try `sierptri sierptri_example` for fun.

Task 1.5 (0 pts).[Bonus] Define a function

```
sierpcarpet : int -> shape
```

that constructs the Sierpinski carpet, whose first few instances are (not to scale)



Task 1.6 (0 pts).[Bonus] Draw something else! Explain your construction in a comment.