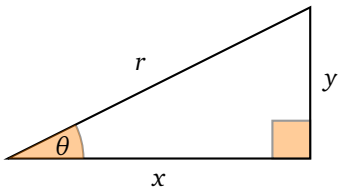
CSE1729 - Introduction to Programming	Due: October 16, 2017
Problem Set 5	

1. There are two main systems of defining points on a two-dimensional plane. One consists of a distance from the origin and an angle from the positive x-axis, referred to as polar coordinates. The other, more familiar system, consists of two components corresponding to the distance along the x-axis and the distance along the y-axis from the origin, referred to as Cartesian coordinates.



   (a) Converting to polar coordinates: Define a SCHEME function named `(c->p p)` which accepts a point in the Cartesian coordinate system as a *pair* and returns another pair representing the same point in the polar coordinate system. That is, if the function receives the pair `(x . y)` as a parameter, it should evaluate to the pair `(r . θ)`, where $r = \sqrt{x^2 + y^2}$ and $\theta = tan^{-1}(\frac{y}{x})$. Your function should take a SCHEME pair as a parameter and return a SCHEME pair. Note: the arctan function in SCHEME is named `atan`.

   (b) Converting to Cartesian coordinates: Define a SCHEME function named `(p->c p)` which accepts a point in the polar coordinate system as a *pair* and returns another pair representing the same point in the Cartesian coordinate system. That is, if the function receives the pair `( r . θ)` as a parameter, it should evaluate to the pair `(x . y)`, where $x = r \cdot cos(\theta)$ and $y = r \cdot sin(\theta)$. Your function should take a SCHEME pair as a parameter and return a SCHEME pair.

2. Let f and g be two functions taking numbers to numbers. We define $\mathbf{m}_{fg}$ to be the function so that

$$\mathbf{m}_{fg}(x) = \text{the larger of } f(x) \text{ and } g(x).$$

   For example, if $f(x) = x$ and $g(x) = -x$ then $\mathbf{m}_{fg}$ is the absolute value function. Define a function `max-f` which takes a *pair* of functions, `f` and `g`, as an argument and returns the function $\mathbf{m}_{fg}$ as its value. (So the return value is a *function*: the function which, at every point x, returns the larger value of $f(x)$ and $g(x)$.)

3. Define a SCHEME function, `zip`, which takes as arguments two lists $(a_1 \ldots a_n)$ and $(b_1 \ldots b_n)$ of the same length, and produces the list of pairs $((a_1.b_1) \ldots (a_n.b_n))$.

4. Define a SCHEME function, `unzip`, which takes a list of pairs $((a_1.b_1) \ldots (a_n.b_n))$ and returns a *pair* consisting of the two lists $(a_1 \ldots a_n)$ and $(b_1 \ldots b_n)$.

   (Note that these functions are not exactly inverses of each other, since `zip` takes its lists as a two arguments, while `unzip` produces a pair of lists.)

5. (**Pairing function.**) A *pairing function* p is a function that places the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ into one-to-one correspondence with the set of all *pairs* of natural numbers (usually denoted $\mathbb{N} \times \mathbb{N}$). It is somewhat surprising that such a function should exist at all: it shows that the set of natural numbers has the same "size" as the set of all *pairs* of natural numbers. To be more precise, a pairing function p takes two natural numbers x and y as arguments and returns a single number z with the property that the original pair can always be determined exactly from the value z. (In particular, the function maps no more than one pair to any particular natural number.)

One pairing function, introduced by Hopcroft and Ullman [1], is the following:

$$p(x,y) = \frac{1}{2}(x+y-2)(x+y-1)+x\,.$$

(a) Write a SCHEME function `encode` that computes the pairing function above. (It should take a pair of numbers as arguments and return a single number.)

(b) As mentioned above, this function has the property that if $(x,y) \neq (x',y')$ then $p(x,y) \neq p(x',y')$: it follows that, in principle, the pair (x,y) can be reconstructed from the single value $z = p(x,y)$. In fact, the values x and y can be reconstructed from $z = p(x,y)$ by first computing the quantities

$$w = \left\lfloor \sqrt{2z} - \frac{1}{2} \right\rfloor, \quad \text{and}$$
$$t = \frac{w^2+w}{2}\,.$$

Then $x = z - t$ and $y = w - x + 2$.

Write a SCHEME function `decode` that takes as an argument an integer z and produces the pair (x,y) for which $p(x,y) = z$. You'll need the `floor` function: `(floor x)` returns the largest integer less than or equal to `x` (that is, it rounds `x` down to the nearest integer).

Hint: You may wish to use the `let*` form for this problem. `let*` has the form

```
(let* ((x1 <expr1>)
       (x2 <expr2>)
       ...
       (xk <exprk>))
  <let-expr>)
```

The form is a simple method for writing "nested `let`s." It is evaluated, informally, as follows. Starting with the external environment, `expr1` is evaluated and the variable `x1` is immediately bound to the resulting value. Following this, `<expr2>` is evaluated in the resulting environment and the variable `x2` is bound to the value. This continues for the remaining variable/expression pairs. Finally, `<let-expr>` is evaluated in the resulting environment and its value is returned. Note, for example, that `x1` may appear in `<expr2>`. (Recall that in a regular `let` expression, the `<expri>` are all evaluated in the external environment.)

6. Write a SCHEME function `positives` which takes a list—call it ℓ—as an argument and returns a list consisting of all elements of ℓ that are positive. In particular, once you have `positives` defined correctly, you should be able to reproduce the following behavior.

```
> (positives (list -2 -1 0 1 2))
'(1 2)
> (positives (list 2 1 0 -2 -1))
'(2 1)
> (positives '(3 1 -1 1 -1))
'(3 1 1)
```

To keep things simple, it's fine if your function just removes from the list all numbers that are zero or less (leaving duplicates in the remaining list, as shown above).

7. Write a SCHEME function, named `remove-duplicates`, that removes all duplicates from a list. (Hint: you might start by defining a function which removes all duplicates of a particular given value v from a list; then what?)

[1] p. 169, *Introduction to Automata Theory, Languages and Computation*, 1979.

Submitting Solutions

Once you have finished:

1. Save your work to a file using the appropriate filename, problemSet5.rkt.
2. Submit your solution file for grading via the MIMIR site.