

MAU11601 Assignment 0
Due 2026-10-01

1. Rewrite the following expressions, written in infix notation, with the precedence and associativity rules which are customary in mathematics, in a form suitable for Racket, i.e. as fully parenthesised prefix expressions. Note that the exponentiation function in Racket is named `expt`.

(a) $\frac{1+2-3}{4-5+6}$

Solution: `(/ (- (+ 1 2) - 3) (+ (- 4 5) 6))`

(b) $1 + 2 < 3^{4^5}$

Solution: `(< (+ 1 2) (expt 3 (expt 4 5)))`

2. Our maze example doesn't respect Newton's first law of motion: that bodies at rest or in motion remain at rest or in motion unless acted upon by an outside force. Change it so that it does. In other words, pressing keys should change the velocity of the catgirl, not just move her by one step.

Solution:

First we need to change the data-type for our world, so that it now has a position and a velocity, and change our start state correspondingly:

```
(define-struct world (position velocity))
(define start (make-world (make-posn half-size half-size)
                           (make-posn 0 0)))
```

The decision to use `posns` for both be position and velocity is debatable. We could have defined a new velocity structure. The advantage is we'd get some extra error checking for free, generating an error if we tried to use a position as a velocity or vice versa. The disadvantage is a bit of extra complexity. In a longer example I'd probably play it safe and define a new structure but here it's not worth the bother.

Next we need to change our `draw` function:

```
(define (draw w)
  (place-image character-cat-girl
               (posn-x (world-position w))
```

```
(posn-y (world-position w))
background))
```

This is essentially the same as our old function but we had to replace `p` by `(world-position w)`, since the argument of our function is now a `world` structure rather than a `posn`.

Next we need to change our `move` function, the one which responded to keyboard input. It's a good idea to rename it as well, since it's no longer moving the catgirl but just changing her velocity. I'll call it `key-update`, to indicate that it's handling keyboard input. The function to handle clock ticks, to be written later, will then be called `clock-update`. The names don't affect the behaviour, as long as we give the names we've chosen to `big-bang`.

Previously we wrote the updated state for each key press with a `make-posn`. We could do that again with `make-worlds` and then a pair of `make-posns` inside but that's ugly and gets very repetitive. Instead we can define a couple of auxiliary functions.

```
(define (update-posn p h k)
  (make-posn (+ h (posn-x p)) (+ k (posn-y p))))
(define (update-velocity w h k)
  (make-world (world-position w)
              (update-posn (world-velocity w) h k)))
(define acceleration 1)
(define (key-update w k)
  (cond
    [(key=? k "w") (update-velocity w 0 (- acceleration))]
    [(key=? k "s") (update-velocity w 0 acceleration)]
    [(key=? k "a") (update-velocity w (- acceleration) 0)]
    [(key=? k "d") (update-velocity w acceleration 0)]
    [else w])))
```

The `update-posn` function shifts a `posn` by specified amounts in the horizontal and vertical direction. It could have been written to take those amounts as a single `posn` rather than as two separate numbers, but the code ends up a bit uglier if you do that, with lots of extra constructors and selectors floating around.

Now we need the `clock-update` function mentioned earlier. Having `update-posn` at our disposal makes this fairly easy.

```
(define (clock-update w)
  (make-world (update-posn (world-position w)
                           (posn-x (world-velocity w))
                           (posn-y (world-velocity w)))
              (world-velocity w)))
```

The only thing left to do is to change the call to `big-bang` to use our new functions.

```
(big-bang start (name "maze")
  (to-draw draw)
  (on-key key-update)
  (on-tick clock-update))
```