Let's add **type inference** to our compiler

Hindley Milner Type Infer

- add/sub num
- equal?, ≤
- vec-set
- not-a-fun
- arity stuff

Wed 5/22 GC

ONE OF { — TYPE INFERENCE / — Reg-Alloc

MIDTERM

5/31 FRIDAY

Example 1

```
(defn (incr x) (+ x 1))
```

(→ (int) int) int

func int int int

```
(incr input)
```

(vec 10 true)

t ::= int
| bool
| (→ (t1 ... tn) t)
| vec t1 t2
| a, b, c ...

mono-types

| forall (a1 ... an) t. poly-type

Example 2

forall (a) (→ (a) a)

```
(defn (id x) x)
```

t1

fn id⟨A⟩(x:A) → A { x }

```
(let* ((a1 (id 7))
       (a2 (id true)))
```

~~true~~) (+ 10 a2)

~~(vec a1 a2)~~

(+ 10 a1)

(→ (int) int)

(forall (a) (→ (a) a))

→ (int) int
→ (bool) bool
→

```
(defn (add x y z)
  (+ x (+ y z)))
```

(→ (int int int) int)

Example 3

forall (a) (→ (→ (a) int) a) int)

```
(defn (f it x)
 (+ (it x) 1))
```

(→ (→ (a) int) a) int)

```
(defn (incr z)
 (+ z 1))

(f incr 10)
```

(if (= input 3)
10
true)

```
(defn (loss b x y)
  (if b x y))
```

(forall (a) (→ (bool a a) a))

Example 4

```
;; --- an API for lists -------------------------------------
(defn (nil) (as (forall (a) (-> () (list a))))
 false)

(defn (cons h t) (as (forall (a) (-> (a (list a)) (list a))))
 (vec h t))

(defn (head l) (as (forall (a) (-> ((list a)) a)))
 (vec-get l 0))

(defn (tail l) (as (forall (a) (-> ((list a)) (list a))))
 (vec-get l 1))

(defn (isnil l) (as (forall (a) (-> ((list a)) bool)))
 (= l false))

;;--- computing with lists ---------------------------------

(defn (length xs)
 (if (isnil xs)
   0
   (+ 1 (length (tail xs)))))

(defn (sum xs)
 (if (isnil xs)
   0
   (+ (head xs) (sum (tail xs)))))

(let (xs  (cons 10 (cons 20 (cons 30 (nil)))))
 (vec (length xs) (sum xs)))
```