Representation Decisions

+ == /

```
data Value:
  | v-num( ____________ )
  | v-bool( ____ )
  | v-clos( ____ )
```

v-num → Number (arbitrary precision rat's | approx doubles)

int32

doubles

v-bool → Boolean

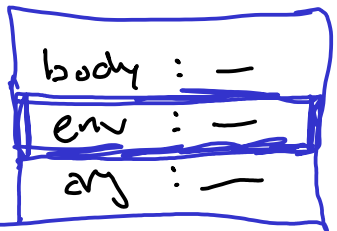
int

"true" / "false"

v-clos → (Env, arg, body)

→ (f :: Function) _ _ _ _ _ _ _

```
fun interp(e::Expr, env::Env):
  | e-lam(arg, body) =>
    v-clos(lam(v :: Value): interp(body, env.set(arg, v)) end)
```



v-clos(env, arg, body)

```
| e-app(func, arg) =>
    fv = interp(func, env)
    av = interp(arg, env)
    fv.f(av)
```

```
type Store = {a: Array<Value>,
              cur-ptr: Number}
```

Looks like real memory...

- alloc — check overflow, then alloc
- deref — store.a[loc]
- update :: Number, Store, V → — store.a[loc] = v

o = { visit-num: lam() ... end,
visit-plus: lam() ... end }

⇒

[dict: "visit-num",
lam() ... end,
"visit-plus",
lam(): ... end]

o.visit-num ⇒ o.get("visit-num")

* errors

o.get(<expr>)

o["visit-num"] ⇒ o.get("visit-num")

expression
position

```
fun getf(obj, fld::Str):
  obj["visit-" + fld]
end
```

⟶ o.get("visit-" + fld)

```
type-of(o.x)
    ot = type-of(o)
      └── {x::Num, y::Num, z::Bool}
```

```
type-of(o[e])
    ot = type-of(o)
    et = type-of(e)
    if et is a regex ...
        do regex matching on ot's fields
```

```
Type
  | regex(re::Regex)
```

"str-const" + e : "str-const(.*)"