UMass Boston Computer Science

**CS450 High Level Languages** (section 2)

# Booleans in “CS450 Lang”

Wednesday, November 13, 2024

# *Logistics*

- HW 10 out
  - due: Mon 11/18 12pm (noon) EST

- Reminder: Pass/Fail & Course Withdraw Deadline
  - Thurs 11/21

Last Time

# Introducing: The "CS450" Programming Lang!

Programmer writes:

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list '+ Expr Expr)
;; - (list '- Expr Expr)
```

parse

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

"eval450"

```
;; A Result is one of:
;; - Number
;; - String
;; - NaN
```

"meaning" of the program

run

(JS semantics)

# *Interlude:* quoting and quasi-quoting

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list '+ Expr Expr)
;; - (list '- Expr Expr)
```

**QUOTING** Shorthand for **constructing S-exprs**

(nested lists of atoms)

```
'(+ 1 2)  ➡  (list '+ 1 2)
```

```
'(+ 1 (+ 2 3))  ➡  (list '+ 1 (list '+ 2 3))
```

**QUASI-QUOTING** Like quoting but **allows "escapes"**

(to "splice in" computed s-exprs)

```
`(+ 1 2)  ➡  (list '+ 1 2)
```

```
`(+ 1 ,(+ 2 3))  ➡  (list '+ 1 5)
```

# “CS450 LANG” Examples

Programmer writes:

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

“eval450”

```
;; A Result is one of:
;; - Number
;; - String
;; - NaN
```

```
(check-equal? (eval450 100) 100)
```

```
(check-equal? (eval450 “one”) “one”)
```

```
(check-equal? (eval450 ‘(+ 100 200)) 300)
```

```
(check-equal? (eval450 ‘(+ “cs” 450)) “cs450”)
```

```
(check-equal? (eval450 ‘(+ 1 2 3 4))   ???   )
```

```
match: no matching clause for '(+ 1 2 3 4)
```

# Dynamic Errors (e.g, Exceptions)

When **a function argument:**

1. **Comes from arbitrary users**
2. **Has a sufficiently complex data definition**
   - (So that **contracts are not enough to enforce the signature**)

Then **dynamic errors** may be needed

# Parsing: “CS450 Lang” Programs

```
;; parse: Expr -> AST
;; Converts a CS450 Lang surface program to its AST
```

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

```
(define (Expr? s)
  (or (number? s)
      (string? s)
      (cons? s)))
```

```
(define/contract (parse s)
  (-> Expr? AST?)
  (match s
   [(? number?) (num s)]
   [(? string?) (str s)]
   [`(+ ,x ,y)  (add (parse x) (parse y))]
   [`(- ,x ,y)  (sub (parse x) (parse y))]))
```

???

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

???

```
(define (AST? s)
  (or (num? s) (str? s)
      (add? s) (sub? s)))
```

# *Interlude:* Inheritance and "Super" Structs

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)
(struct AST [])
(struct num AST [val])
(struct str AST [val])
(struct add AST [lft rgt])
(struct sub AST [lft rgt])
```

"abstract" struct (implicitly defines `AST?` predicate)

"super" struct declaration

Alternatively ...

```
(define (AST? s)
  (or (num? s) (str? s)
      (add? s) (sub? s)))
```

e.g., if `p = (sub (num 1) (num 2))` then **both** `(sub? p) = true` and `(AST? p) = true`

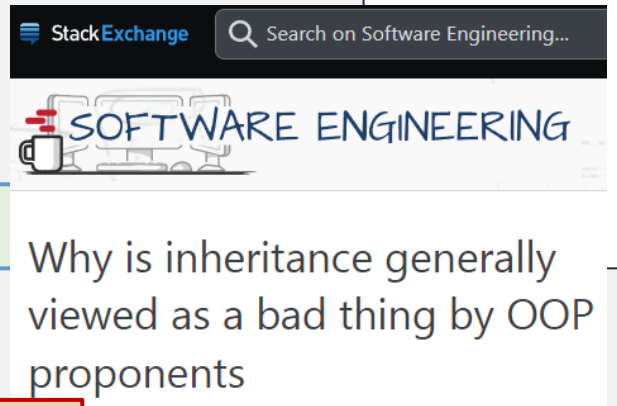
# *Interlude:* Inheritance and "Super" Structs

This kind of "polymorphic" "abstract" data definition is what we've been creating all semester!

"super" structs are just a convenience for the same thing (when all itemizations are structs)

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)
(struct AST [])
num AST [val])
str AST [val])
add AST [lft rgt])
sub AST [lft rgt])
```

"abstract" struct (implicitly defines AST? predicate)




WAIT, I heard "Inheritance is bad"???

NO, accepted OO principles says:

Inheritance of implementations is bad ☒ (violates "1 task, 1 function")

Interfaces and abstract classes are ok ☑ (i.e., "itemizations")

# Parsing: “CS450 Lang” Programs

```
;; parse: Expr -> AST
;; Converts a CS450 Lang surface program to its AST
```

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

```
(define/contract (parse s)
  (-> Expr? AST?)
  (match s
   [(? number?) (num s)]
   [(? string?) (str s)]
   [`(+ ,x ,y)  (add (parse x) (parse y))]
   [`(- ,x ,y)  (sub (parse x) (parse y))]
   [_ (error … )]))
```

function argument:

1. Comes from arbitrary users
2. Has sufficiently complex data definition where contracts are insufficient

# *Interlude:* Racket exceptions

Exceptions are just special structs

Super struct (enables using **exception API**)

```
(struct exn:fail:syntax:cs450 exn:fail:syntax [])
```

```
(define/contract (parse s)
  (-> Expr? AST?)
  (match s
   [(? number?) (num s)]
   [(? string?) (str s)]
   [`(+ ,x ,y)  (add (parse x) (parse y))]
   [`(- ,x ,y)  (sub (parse x) (parse y))]
   [_ (raise-syntax-error
        'parse "not a valid CS450 Lang program" s
        #:exn exn:fail:syntax:cs450)]))
```

# “CS450 Lang” Examples

Programmer writes:

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

“eval450”

```
;; A Result is one of:
;; - Number
;; - String
;; - NaN
```

```
(check-equal? (eval450 ‘(+ 1 2 3 4))  ???  )
```

~~match: no matching clause for '(+ 1 2 3 4)~~

```
parse: not a valid CS450 Lang program in: (+ 1 2 3 4)
```

```
(check-exn exn:fail:syntax:cs450?
    (λ () (eval450 ‘(+ 1 2 3 4)))
```

# HW10

- **Add some boolean constructs** (also follows JS semantics):
  - TRUE/ FALSE literal values
  - **“equality” === comparator** (look up JS ===)
  - **“Ternary conditional” expression,** with **“truthiness” test** (look it up)

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - ‘TRUE
;; - ‘FALSE
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
;; - (list ‘=== Expr Expr)
;; - (list Expr ‘? Expr ‘: Expr)
```

# HW10: Data Definitions

- **Refactor data definitions?** (your design choice)

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - (list '+ Expr Expr)
;; - (list '- Expr Expr)
;; - (list '=== Expr Expr)
;; - (list Expr '? Expr ': Expr)
```

```
;; A Atom is one of:
;; - Number
;; - String
;; - 'TRUE
;; - 'FALSE
```

**or**

```
;; A Atom is one of:
;; - Number
;; - String
;; - SymBool (hah)
```

```
0LangExpr (Expr) is one of:
mber
;; - String
;; - 'TRUE
;; - 'FALSE
'+ Expr Expr)
'- Expr Expr)
Expr)
```

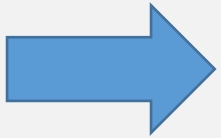
```
;; A SymBool is one of:
;; - 'TRUE
;; - 'FALSE
```

# HW10: Data Definitions

- **Refactor data definitions?** (your design choice)

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```



**???**

I will collect so all can use for hw (even if it's wrong!)

# In-class Coding: **write your own GradeScope tests!**

```
;; parse: Expr -> AST
;; Parses "CS450 Lang" program to AST
```

```
;; run: AST -> Result
;; Computes result of running CS450 AST
```

```
(define eval450 (compose run parse))
```

```
;; A 450LangExpr (Expr) is one
;; - Number
;; - String
;; - 'TRUE
;; - 'FALSE
;; - (list '+ Expr Expr)
;; - (list '- Expr Expr)
;; - (list '=== Expr Expr)
;; - (list Expr '? Expr ': Expr)
```

```
;; "adding" bools
(check-equal? (eval450 '(+ TRUE FALSE)) 1)
```

```
;; equality
(check-true (eval450'(=== (+ TRUE FALSE) 1)))
```

```
;; js "truthy true"
(check-equal? (eval450 '(10 ? 100 : 200)) 100)
;; js "truthy false"
(check-equal? (eval450 '((- 100 100) ? "a" : "b")) "b")
```

```
;; - Number
;; - String
;; - Boolean
;; - NaN
```

- For `===`: lookup JavaScript " strict equality"
- For `?`: lookup JavaScript "truthy" values
- Experiment: `repljs.com/new`

Last Time

# Introducing: The "CS450" Programming Lang!

Programmer writes:

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - (list '+ Expr Expr)
;; - (list '- Expr Expr)
```

"eval450"

```
;; A Result is one of:
;; - Number
;; - String
;; - NaN
```

Next Feature: **Variables?**

parse

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

run

(JS semantics)

NOTE: not needed for hw10

# Adding Variables

```
;; A Variable is a Symbol
```

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list '+ Expr
;; - (list '- Expr
```

```
;; A Result is
;; - Number
;; - String
;; - NaN
```

parse

run

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (var Symbol)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct var [name])
(struct add [lft rgt])
```

**$Q_1$**: What is the "meaning" of a variable?

**$A_1$**: Whatever "value" it is bound to

**$Q_2$**: Where do these "values" come from?

**$A_2$**: Other parts of the program

The run function needs to "remember" these values ( with an **accumulator**!)

# `run450js`, with an accumulator

```
;; run: AST -> Result
;; Computes result of running CS450 AST
(define (run p)
  ;; accumulator acc : Environment
  ;; invariant: Contains variable+result pairs that are currently in-scope
  (define (run/acc p acc)
    (match p
     [(num n) n]
     [(add x y) (450+ (run x) (run y))]))
 (run/acc p  ???  ))
```

# Environments

- A data structure that "associates" two things together
  - E.g., maps, hashes, etc
  - For simplicity, let's use list-of-pairs

```
;; An Environment is one of:
;; - empty
;; - (cons (list Var Result) Environment)
```

```
;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450-lang variables
```

```
;; if there are duplicates,
;; vars at front of list shadow those in back
```

# Environments

- A data structure that “associates” two things together
  - E.g., `maps`, `hashes`, etc
  - For simplicity, let’s use list-of-pairs
- Needed operations:
  - `add : Env Var Result -> Env`
  - `Lookup : Env Var -> Result`

# run, with an Environment

**TODO:**
- When are variables “added” to environment
- Initial environment?

```
;; run: AST -> Result
;; Computes result of running CS450 Lang AST
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable+result pairs that are in-scope
  (define (run/acc p env)
    (match p
     [(num n) n]
     [(var x) (lookup env x)]
     [(add x y) (450+ (run x env) (run y env))]))
 (run/acc p  ???  ))
```

NOTE: not needed for hw10

# Adding Variables

```
;; A 450LangExpr (Expr) is one of:
;; - Number
;; - String
;; - Variable
;; - (list ‘bind [Variable Expr] Expr)
;; - (list ‘+ Expr Expr)
;; - (list ‘- Expr Expr)
```

parse

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (var Symbol)
;; - (bind Symbol AST AST)
;; - (add AST AST)
;; - (sub AST AST)

(struct num [val])
(struct str [val])
(struct var [name])
(struct bind [var expr body])
(struct add [lft rgt])
(struct sub [lft rgt])
```

# run, with an Environment

```
;; run: AST -> Result
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable:result pairs that are in scope
  (define (run/acc p env)
    (match p
     [(num n) n]
     [(var x) (lookup env x)]
     [(bind x e body) (run body (add env x (run e env)))]
     [(add x y) (450+ (run x env) (run y env))]))
 (run/acc p  ???  ))
```

3. run body with that new environment

2. add variable x to environment

1. Compute Result that variable x represents

# In-class Coding 11/13: **write env ops**

- Needed operations:
  - `add : Env Var Result -> Env`
  - `Lookup : Env Var -> Result`

```
;; An Environment is one of:
;; - empty
;; - (cons (list Var Result) Environment)
```

```
;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450-lang variables
```

```
;; if there are duplicates,
;; vars at front of list shadow those in back
```

Think about examples where this happens!