Logistics

• HW 7 out
  • due: Sun 11/19 11:59 pm EST
    • Really due: Wed 11/22 11:59 pm EST

• 2 submissions 😊

• (no hw over Thanksgiving)
HW 5 (Pong Game) recap

Key Points

- Think about data organization
  - As World gets more components, need subgroupings of data
  - Be careful with: extraneous data, e.g., does player need x data?

```plaintext
;; A WorldState is a:
;; (world x y vx vy p1 p2 sc1 sc2)
;; x   : XCoordinate
;; y   : YCoordinate
;; vx  : Velocity
;; vy  : Velocity
;; p1  : YCoordinate
;; p2  : YCoordinate
;; sc1 : NaturalNum in [0,10]
;; sc2 : NaturalNum in [0,10]
```

```
;; A WorldState is a
;; (world b p1 p2 sc1 sc2)
;; b   : Ball
;; p1  : YCoordinate
;; p2  : YCoordinate
;; sc1 : NatNum in [0,10]
;; sc2 : NatNum in [0,10]
```

or

```
;; A WorldState is a
;; (world b p1 p2)
;; b   : Ball
;; p1  : Player
;; p2  : Player
```
HW 5 (Pong Game) recap

Key Points
• Think about data organization
• Data organization affects code organization (and readability)

;;; A WorldState is a
;;; (world b p1 p2)
;;; b : Ball
;;; p1 : Player
;;; p2 : Player

;;; next-world : WorldState -> WorldState
(define (next-world w)
  (match-define (world b p1 p2) w)
  (world (if (scored? b)
              (mk-center-ball/rand-val)
              (next-ball b p1 p2))
          (update p1 score b p1)
          (update p2 score b p2)))
HW 5 (Pong Game) recap

Key Points

- Think about data organization
- Data organization affects code organization (and readability)

Think about fn names! “ball-in-scene/x?” no longer makes sense!!!

```scheme
;; A WorldState is a
;; (world b p1 p2)
;; b : Ball
;; p1 : Player
;; p2 : Player

;; next-world : WorldState -> WorldState
(define (next-world w)
  (match-define (world b p1 p2) w)
  (world (if (scored? b)
    (mk-center-ball/rand-val)
    (next-ball b p1 p2)))

... Update player1 ...
... Update player2 ...
```
Don’t submit this kind of code in this class (or in software engineering)

Which is more readable (by humans)?

60 lines

70 lines

80 lines
HW 5 (Pong Game) recap

Key Points
• Think about data organization
• Data organization affects code organization (and readability)
• Break up large data defs into (logical) smaller, readable ones
• Break up large functions into (logical) smaller, readable ones
HW 6 (editor + mouse) Recap

Key Points

- Designing functions with accumulators

```scheme
;; mouse-handler : Editor Coord Coord MouseEvt -> Editor
(define (mouse-handler ed x y mevt)
  (cond
   [(mouse=/ mevt "button-down") (split ed x)]
   [else ed]))
```

```
new pre initial (chars-sofar):
  empty (no chars)

new post initial (rest-chars):
  (ed->list ed0) (all chars)
```

```
Editor (new)

new pre | new post
```

```
Invariant:
  "so far" + "remaining" = ed0
```

```
Need to compute new split (at mouse x)
```

```
Editor ed0 (current)

pre | post
```
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
   ;; split/a : List<1str> List<1str> Coord -> Editor
   ;; ACCUMULATOR: chars-so-far
   ;; invariant: represents (rev) of chars seen so far, where
   ;;   (append (rev chars-so-far) rest-chars) = (ed->1st ed0)
   ;;   and: (image-width (render-chars chars-so-far)) < x
   (define (split/a chars-sofar rest-chars x)
      (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
            [else (match-define (cons c rst) rest-chars)
                (if (chars-at-cursor? x (cons c chars-sofar))
                    (editor (cons c chars-sofar) rst)
                    (split/a (cons c chars-sofar) rst x))])
   (split/a empty (ed->1st ed0) x))
```
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;   and: (image-width (render-chars chars-so-far)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [[(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else (match-define (cons c rst) rest-chars)
                (if (chars-at-cursor? x (cons c chars-sofar))
                    (editor (cons c chars-sofar) rst)
                    (split/a (cons c chars-sofar) rst x))]])
    (split/a empty (ed->lst ed0) x))

new pre initial (chars-sofar): empty (no chars)
new post initial (rest-chars): (ed->lst ed0) (all chars)


HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

(define (split ed0 x) ;; split : Editor Coord -> Editor
   ;; split/a : List<1str> List<1str> Coord -> Editor
   ;; ACCUMULATOR: chars-so-far
   ;; invariant: represents (rev) of chars seen so far, where
   ;;   (append (rev chars-so-far) rest-chars) = (ed->1st ed0)
   ;;   and: (image-width (render-chars chars-so-far)) < x
   (define (split/a chars-sofar rest-chars x)
      (cond [[(empty? rest-chars) (editor chars-sofar rest-chars)]
         [else       (match-define (cons c rst) rest-chars)
         (if (chars-at-cursor? x (cons c chars-sofar))
            (editor (cons c chars-sofar) rst)
            (split/a (cons c chars-sofar) rst x))]]
      (split/a empty (ed->1st ed0) x))

new pre initial (chars-sofar): empty (no chars)

new post initial (rest-chars): (ed->list ed0) (all chars)

Invariant: “so far” + “remaining” = ed0
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

```scheme
(define (split ed0 x) ;; split : Editor Coord -> Editor
    ;; split/a : List<1str> List<1str> Coord -> Editor
    ;; ACCUMULATOR: chars-so-far
    ;; invariant: represents (rev) of chars seen so far, where
    ;; (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
    ;; and: (image-width (render-chars chars-so-far)) < x
    (define (split/a chars-sofar rest-chars x)
      (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
            [else (match-rest-char (cons c rst) rest-chars)
                   (if (chars-at-cursor? x (cons c chars-sofar))
                       (editor (cons c chars-sofar) rst)
                       (split/a (cons c chars-sofar) rst x))])
      (split/a empty (ed->lst ed0) x))
```

HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

```scheme
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;; (append (rev chars-so-far) rest-chars) = (ed->1st ed0)
  ;; and: (image-width (render-chars chars-so-far)) < x
  (define (split/a chars-so-far rest-chars x)
    (cond [(empty? rest-chars) (editor chars-so-far rest-chars)]
          [else (match-define (cons c rst) rest-chars)
               (if (chars-at-cursor? x (cons c chars-so-far))
                   (editor (cons c chars-so-far) rst)
                   (split/a (cons c chars-so-far) rst x))]]
    (split/a empty (ed->1st ed0) x))
```

```scheme```
```
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

(define (split ed0 x) ;; split : Editor Coord -> Editor
 ;; split/a : List<1str> List<1str> Coord -> Editor
 ;; ACCUMULATOR: chars-so-far
 ;; invariant: represents (rev) of chars seen so far, where
 ;;   (append (rev chars-so-far) rest-chars) = (ed->1st ed0)
 ;;       and: (image-width (render-chars chars-so-far)) < x
    (define (split/a chars-sofar rest-chars x)
      (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
            [else (match-define (cons c rst) rest-chars)
                   (if (chars-at-cursor? x (cons c chars-sofar))
                       (editor (cons ??? chars-sofar) rst)
                       (split/a (cons c chars-sofar) rst x))])
      (split/a empty (ed->1st ed0) x))
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

(define (split ed0 x) ;; split : Editor Coord -> Editor
;;;; split/a : List<1str> List<1str> Coord -> Editor
;;;; ACCUMULATOR: chars-so-far
;;;; invariant: represents (rev) of chars seen so far, where
;;;; (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
;;;; and: (image-width (render-chars chars-sofar)) < x
(define (split/a chars-sofar rest-chars x)  
  (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]  
    [else (match_DEFINE (cons c rst) rest-chars)  
      (if (chars-at-cursor?) x (cons c chars-sofar))  
      (editor (cons c chars-sofar) rst)  
      (split/a (cons c chars-sofar) rst x))])  
(split/a empty (ed->lst ed0) x))
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

```
(define (split ed0 x) ;; split : Editor Coord -> Editor
  ;; split/a : List<1str> List<1str> Coord -> Editor
  ;; ACCUMULATOR: chars-so-far
  ;; invariant: represents (rev) of chars seen so far, where
  ;;   (append (rev chars-so-far) rest-chars) = (ed->lst ed0)
  ;;   and: (image-width (render-chars chars-so-far)) < x
  (define (split/a chars-sofar rest-chars x)
    (cond [(empty? rest-chars) (editor chars-sofar rest-chars)]
          [else (match-define (cons c rst) rest-chars)
                (if (chars-at-cursor? x (cons c chars-sofar))
                    (compute-before/after x chars-sofar c rst)
                    (split/a (cons c chars-sofar) rst x))]]
  (split/a empty (ed->lst ed0) x))
```
HW 6 (editor + mouse) Recap

Key Points

• Designing functions with accumulators

• Code should be sufficiently “readable” such that ...
  • You can present your code like I just did!
  • (Still on the table for this semester ...)

18
Introducing: The “CS450js” Programming Lang!

Last Time

Programmer writes:

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

Next Feature: Variables?

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
```

“eval450js”

```plaintext
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
```

“meaning” of the program

`parse450js`

Last Time

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

`run450js` (JS semantics)
Adding Variables

;;; A Variable is a Symbol

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

;;; A 450jsAST is one of:
;;; - (num Number)
;;; - (str String)
;;; - (var Symbol)
;;; - (add 450jsAST 450jsAST)
;;; - (sub 450jsAST 450jsAST)

;;; A 450jsResult is one of:
;;; - Number
;;; - String
;;; - NaN

---

Q₁: What is the “meaning” of a variable?
A₁: Whatever “value” it is bound to

Q₂: Where do these “values” come from?
A₂: Other parts of the program

The run function needs to “remember” these values (with an accumulator)
run450js, with an accumulator

;; run: 450jsAST -> 450jsResult
;; Computes result of running a CS450js program 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/acc x) (run/acc 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 450jsResult) Environment)

;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450js-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:
  • env-add : Env Var Result -> Env
  • env-lookup : Env Var -> Result
In-class Coding 11/21: write env ops

• Needed operations:
  • env-add : Env Var 450jsResult -> Env
  • env-lookup : Env Var -> 450jsResult

;; An Environment (Env) is one of:
;; - empty
;; - (cons (list Var 450jsResult) Environment)

;; interpretation: a runtime environment for
;; (ie gives meaning to) cs450js-lang variables

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

Think about examples where this happens!
run450js, with an Environment

;; run: 450jsAST -> 450jsResult
;; Computes result of running CS450js 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/acc x env) (run/acc y env))]]
    (run/acc p ??? )))
Programs that Create Variables

```plaintext
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list `bind Var 450jsExpr 450jsExpr)
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list `- 450jsExpr 450jsExpr)
```

```plaintext
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (var Symbol)
;; - (bind Symbol 450jsAST 450jsAST)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)

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

parse450js
run450js, with an Environment

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

1. Compute 450jsResult that variable x represents
2. add variable x to environment
3. run body with that new environment
(define (run p)
  ;; accumulator env : Environment
  ;; invariant: Contains variable+result pairs that are in-scope
  (define (run/e p env)
    (match p
      [(num n) n]
      [(var x) (lookup env x)]
      [(bind x e body) (run/e body (add env x (run/e e env)))]
      [(add x y) (450+ (run/e x env) (run/e y env)))]
    (run/e p ???)
  ))
A 450jsExpr is one of:
- Number
- String
- Variable
- (list `bind Var 450jsExpr 450jsExpr)
- (list `+ 450jsExpr 450jsExpr)
- (list `- 450jsExpr 450jsExpr)

Can these go into initial env?

**Next Feature: Functions?**

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

**Should these be different language constructs?**
Initial Environment

A 450jsExpr is one of:
- Number
- String
- Variable
- (list ‘bind Var 450jsExpr 450jsExpr)
- (list ‘+ 450jsExpr 450jsExpr)
- (list ‘- 450jsExpr 450jsExpr)
- (list ‘fn List<Variable> 450jsExpr)

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

Next Feature: Functions?
Initial Environment

### A 450jsExpr is one of:
- Number
- String
- Variable
- (list 'bind Var 450jsExpr 450jsExpr)
- (list '+' 450jsExpr 450jsExpr)
- (list '-' 450jsExpr 450jsExpr)
- (list 'fn List<Variable> 450jsExpr)

### A 450jsResult is one of:
- Number
- String
- NaN
- ???

**TODO:**
- When are variables “added” to environment?
- Initial environment?
- Function representation?
- Function result?
Initial Environment

`; A 450jsExpr is one of:
`; - Number
`; - String
`; - Variable
`; - (list `bind Var 450jsExpr 450jsExpr)
`; - (list `+ 450jsExpr 450jsExpr)
`; - (list `* 450jsExpr 450jsExpr)
`; - (list `fn List<Variable> 450jsExpr)
`; - (list `fnCall 450jsExpr 450jsExpr)

; A 450jsResult is one of:
`; - Number
`; - String
`; - NaN
`; - ????

TODO:
- When are variables “added” to environment
  - Initial environment?
  - Function representation
  - Function result?
  - Function calls?
### Initial Environment

```plaintext
;; A 450jsExpr is one of:
;;  - Number
;;  - String
;;  - Variable
;;  - (list 'bind Var 450jsExpr 450jsExpr)
;;  - (list '+' 450jsExpr 450jsExpr)
;;  - (list '-' 450jsExpr 450jsExpr)
;;  - (list 'fn List<Variable> 450jsExpr)
;;  - (list 'fncall 450jsExpr 450jsExpr)
```

### TODO:
- When are variables “added” to environment
- Initial environment?
- Function representation
- Function result?
- Function calls?

```plaintext
;; A 450jsResult is one of:
;;  - Number
;;  - String
;;  - NaN
;;  - ????
```
Initial Environment

;; A 450jsExpr is one of:
;; - Number
;; - String
;; - Variable
;; - (list `bind Var 450jsExpr 450jsExpr)
;; - (list `+ 450jsExpr 450jsExpr)
;; - (list `- 450jsExpr 450jsExpr)
;; - (list `fn List<Variable> 450jsExpr)
;; - (list `fncall 450jsExpr 450jsExpr)

Most language function calls don’t explicitly require saying “function call”

TODO:
- When are variables “added” to environment?
- Initial environment?
- Function representation
- Function result?
- Function calls?

;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
No More Quizzes!

but push your in-class work to:
Repo: cs450f23/lecture21-inclass