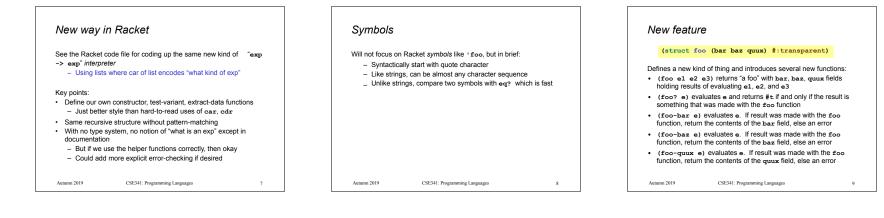
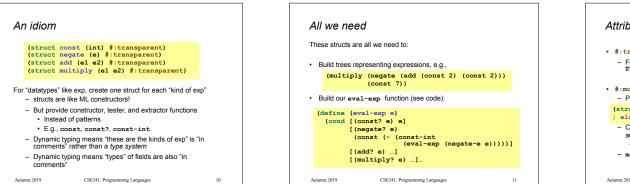


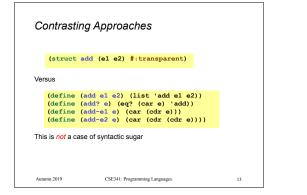
| Recursive structures |
|--|
| More interesting datatype-programming we know: |
| <pre>datatype exp = Const of int</pre> |
| <pre>fun eval_exp e = case e of Const i => i Negate e2 => ~ (eval_exp e2) Add(e1,e2) => (eval_exp e1) + (eval_exp e2) Multiply(e1,e2)=>(eval_exp e1)*(eval_exp e2)</pre> |
| Autumn 2019 CSE341: Programming Languages 5 |

| Change h | ow we do this | |
|--|---|--------|
| Previous vers | sion of eval_exp has type exp -> in | it |
| From now on | will write such functions with type exp | -> exp |
| of results (int | use will be interpreting languages with m s, pairs, functions,) ugh much more complicated for example | · |
| How? See th | e ML code file: | |
| Base case Recursive | e returns entire expression, e.g., (Cons e cases: | st 17) |
| Check | variant (e.g., make sure a Const) | |
| Extrac | t data (e.g., the number under the Cons | st) |
| Also r | eturn an exp (e.g., create a new Const | :) |
| Autumn 2019 | CSE341: Programming Languages | 6 |





| ttributes |
|--|
| #:transparent is an optional attribute on struct definitions For us, prints struct values in the REPL rather than hiding them, which is convenient for debugging homework |
| #:mutable is another optional attribute on struct definitions Provides more functions, for example: |
| <pre>(struct card (suit rank) #:transparent #:mutable) ; also defines set-card-suit!, set-card-rank!</pre> |
| Can decide if each struct supports mutation, with usual advantages and disadvantages As expected, we will avoid this attribute mcons is just a predefined mutable struct |
| tumn 2019 CSE341: Programming Languages 12 |



The key difference

(struct add (e1 e2) #:transparent)

- The result of calling (add x y) is not a list
 And there is no list for which add? returns #t
- struct makes a new kind of thing: extending Racket with a new kind of data
- So calling car, cdr, or mult-e1 on "an add" is a run-time error

14

Autumn 2019 CSE341: Programming Languages

List approach is error-prone (define (add el e2) (list 'add el e2)) (define (add? e) (eq? (car e) 'add))

(define (add? e) (eq? (car e) 'add)) (define (add-el e) (car (cdr e))) (define (add-e2 e) (car (cdr (cdr e))))

- Can break abstraction by using car, cdr, and list-library functions directly on "add expressions"
 Silent likely error:
- (define xs (list (add (const 1)(const 4)) ...))
 (car (car xs))
- Can make data that add? wrongly answers #t to (cons 'add "I am not an add")
- Autumn 2019 CSE341: Programming Languages

15

