



### Why are macros difficult to use sensibly CSE341: Programming Languages · Using Racket's macro system Lecture 16 Defining macros - Watching out for evaluation order and (re)-evaluation Macros - Why hygiene makes Racket's macros much easier to use sensibly Dan Grossman When (not) to use macros Fall 2011 Fall 2011 CSE341: Programming Languages Tokenization What is a macro A macro describes how to transform some new syntax into First question for a macro system: How does it tokenize? different syntax in the source language Macro systems generally work at the level of tokens not A macro is one way to implement syntactic sugar sequences of characters - "Replace any syntax of the form e1 andalso e2 with - So must know how programming language tokenizes text if e1 then e2 else false" • Example: "replace all occurrences of car with hd" A macro system is a language (or part of a larger language) for - Would not rewrite (+ cart foo) to (+ hdt foo) defining macros - Would not rewrite car-door to hd-door • But would in C where car-door is subtraction Macro expansion is the process of rewriting the syntax to eliminate macro uses - Before a program is run (or even compiled) Fall 2011 Fall 2011 CSE341: Programming Languages 3 CSE341: Programming Languages

## Parenthesization

Second question for a macro system: How does associativity work?

C/C++ basic example: #define ADD(x,y) x+y

Probably not what you wanted:

ADD (1,2/3)\*4 means 1+2/3\*4 not (1+2/3)\*4

So C macro writers use lots of parentheses, which is fine:

### #define ADD(x, y) ((x)+(y))

Racket won't have this problem:

– Macro use: (macro-name ...)

- After expansion: ( *something else in same parens* )

Fall	201	1
------	-----	---

5

## Local bindings

This lecture

What are macros

Third question for a macro system: Can variables shadow macros?

Suppose macros also apply to variable bindings. Then:

	•	([hd 0][car 1]) hd) ; 0 ([hd 0][car 1]) hd) ; 0						
Would become:								
	•	([car 0][car 1]) car) ; erros ([car 0][car 1]) car) ; 1	r					

This is why C/C++ convention is all-caps macros and non-all-caps for everything else

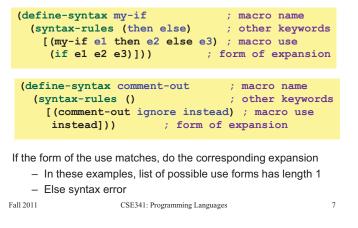
Racket gets this and other scope gotchas "right"

2

4

## Example Racket macro definitions

Two simple macros



# Example uses

It's like we added keywords to our language

- Other keywords only keywords in uses of that macro
- Syntax error if keywords misused
- Rewriting ("expansion") happens before execution

```
(my-if x then y else z) ; (if x y z)
(my-if x then y then z) ; syntax error
(my-if x then (begin (print "hi") 34) then 15)
(comment-out (begin (print "hi") 34) 15)
(comment-out (car null) #f)
```

Fall 2011

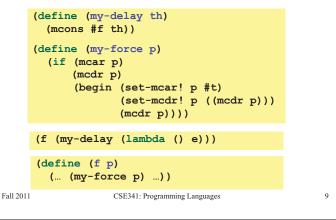
CSE341: Programming Languages

#### 8

### Revisiting delay and force

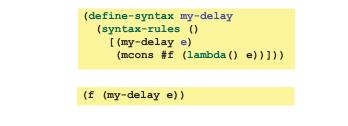
Recall our definition of promises from last lecture

- Should we use a macro instead to avoid clients' explicit thunk?



## A delay macro

- A macro can put an expression under a thunk
   Delays evaluation without explicit thunk
  - Cannot implement this with a function
- Now client then should *not* use a thunk (that would double-thunk)
  - Racket's pre-defined delay is a similar macro



CSE341: Programming Languages

What about a force macro?

We could define my-force with a macro too

- Good macro style would be to evaluate the argument exactly once (use x below, not multiple evaluations of e)
- Which shows it is bad style to use a macro at all here!
- Don't use macros when functions do what you want

# Another bad macro

Any function that doubles its argument is fine for clients

```
(define (dbl x) (+ x x))
(define (dbl x) (* 2 x))
```

- These are equivalent to each other

So macros for doubling are bad style but instructive examples:

(define-syntax	dbl	(syntax-rules()[(dbl	<b>x) (+</b>	х	x)]))
(define-syntax	dbl	<pre>(syntax-rules()[(dbl</pre>	<b>x</b> ) (*	2	x)]))

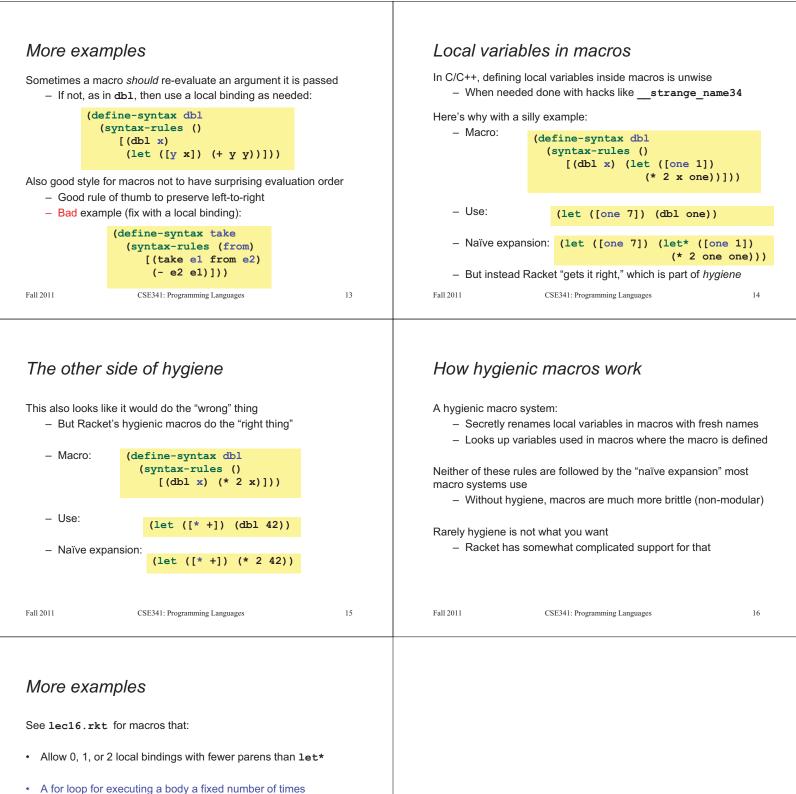
- These are not equivalent to each other. Consider:

(dbl (begin (print "hi") 42))

```
Fall 2011
```

Fall 2011

10



- A for loop for executing a body a fixed number of tim
- A re-implementation of let\* in terms of let
  - Requires macros that take any number of arguments
  - Requires recursive macros