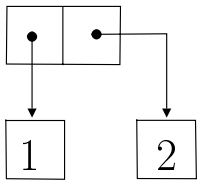
CS320 Handout 11
Pairs and Lists

Three Descriptions of Pairs

Input Expression

```
(cons 1 2)
```

Box and Pointer



Output Representation – using *dotted pair notation*

```
(1 . 2)
```

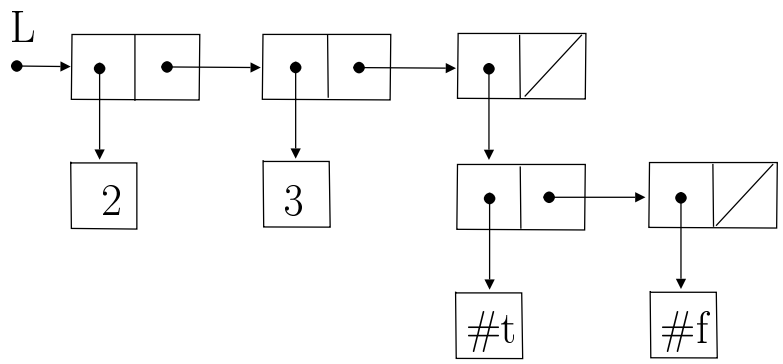
Three Descriptions of Lists

Input Expression

```
(define L (list 2 (+ 1 2) (list #t #f)))
```

Recall that
  (list ⟨v_1⟩ ⟨v_2⟩ ··· ⟨v_n⟩)
is equivalent to
  (cons ⟨v_1⟩ (cons ⟨v_2⟩ (··· (cons ⟨v_n⟩ nil) ···)))

Box and Pointer



Output Representation

```
(2 3 (#t #f))
```

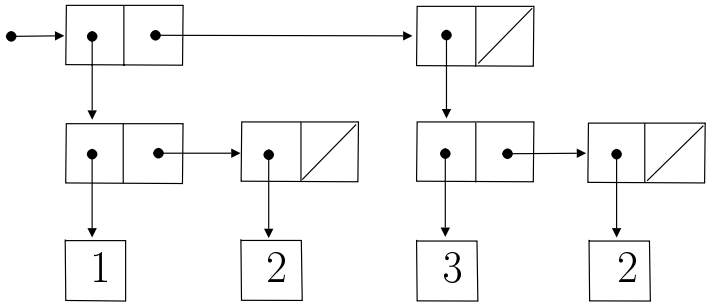
Using dotted pair notation, you may write instead:
  (2 . (3 . (#t . (#f . ()))))
But this is more verbose and less readable!

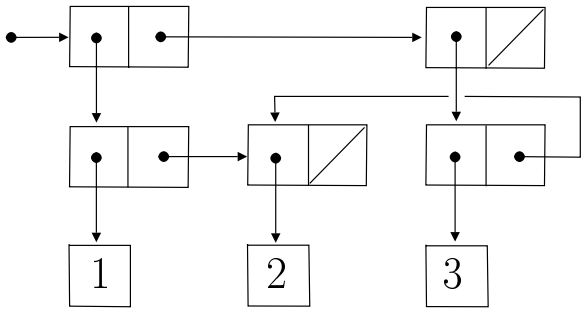
"Box and Pointer" More Informative Than "Output Representation"

Input Expression	Output Representation
`(list (list 1 2)` `  (list 3 2))`	`((1 2) (3 2))`

`(let ((x (list 2)))`
`  (list (cons 1 x)`
`  (cons 3 x)))`
	`((1 2) (3 2))`

Box and Pointer





Several Hand Exercises

```
;; A Scheme function that prints list structures in the same way
;; that a Scheme interpreter does:

(define (print-list-structure1 x)
  (cond ((null? x) (display "()"))
        ((not (pair? x)) (display x))
        (else (display "(")
              (print-list-structure1 (car x))
              (display " . ")
              (print-list-structure1 (cdr x))
              (display ")"))))

;; But the preceding is not the standard way of printing out
;; list structures. The standard way is implemented by the
;; following Scheme function:

(define (print-list-structure2 x)
  (define (print-contents x)
    (display (car x))
    (cond ((null? (cdr x)) () )
          ((not (pair? (cdr x)))
           (display " . ")
           (display (cdr x)))
          (else
           (display " ")
           (print-contents (cdr x)))))
  (cond ((null? x) (display "()"))
        ((not (pair? x)) (display x))
        (else (display "(")
              (print-contents x)
              (display ")"))))
```

```
;; A Scheme expression that prints out (1 2 3) -- yes, very easy:
;; (list 1 2 3)

;; A Scheme expression that prints out (1 2 . 3):
;; Is it (list 1 (cons 2 3)) ? NO
;; Is it (cons 1 (cons 2 3)) ? YES

;; A Scheme expression that prints out (1 . 2 3):
;; Is it (cons 1 (list 2 3)) ? NO
;; Is it (display "(1 . 2 3)") ? YES
;; Is it (list 1 "." 2 3) ? NO
;; Is it (list 1 '. 2 3) ? YES
```