Topic 6
Hierarchical Data and the Closure Property

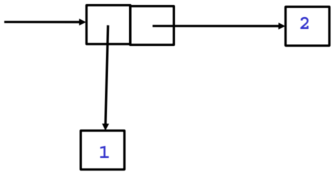
Section 2.2.1

September 2008

Box and pointer notation

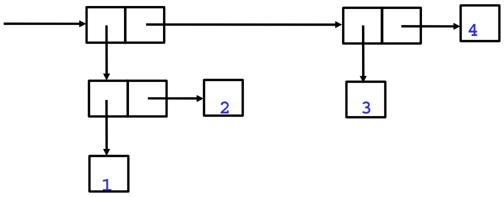
- Draw cdr pointers to the right
- Draw car pointers downward

`(cons 1 2)`



Another list structure

`(cons (cons 1 2) (cons 3 4))`



The closure property

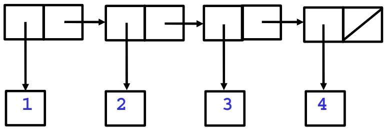
- A constructor has the **closure property if it can take data of a certain type as input and return data of the same type**
- **`cons` is an example**
- **Such constructors can be used to build hiearchical structures**

Lists, a recursive data type

- The empty list is a list
- If `x` is any datum and `y` is a list, then `(cons x y)` **is a list**
- **The empty list is denoted by `empty` in DrScheme and by `nil` in the course textbook**
- **Whenever you see `nil` in the book, read `empty`**

What do lists look like?

```
(cons 1 ( cons 2 (cons 3 (cons 4 empty))))
(1 2 3 4)
```

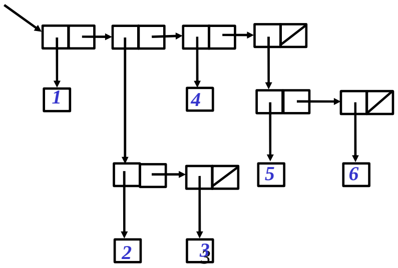


Lists can contain lists

```
(cons 1
      (cons (cons 2 (cons 3 empty))
            (cons 4
                  (cons (cons 5 (cons 6 empty))
                         empty))))

(1 (2 3) 4 (5 6))
```

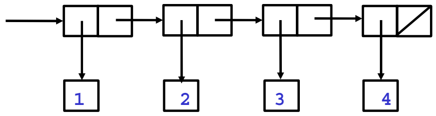
Box and pointer representation



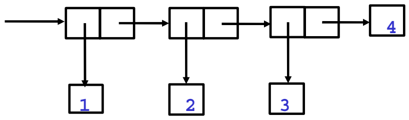
Printing out list structures

Printed like lists, but if the last `cdr` **in a** `cdr` **chain points to a primitive datum other than** `empty`**, the primitive datum is printed with a dot in front of it.**

A comparison



```
(1 2 3 4)
```



```
(1 2 3 . 4)
```

Some service procedures for lists

```
(list 1 2 3 4) --> (1 2 3 4)
```

;; takes a list with at least n elements

;; and returns the nth element of the list

;; note counting starts from 0

```
(define (our-list-ref lst n)
  (if (= n 0)
      (car lst)
      (our-list-ref (cdr lst)
                    (- n 1))))

(our-list-ref (list 1 2 3 4) 0) --> 1
(our-list-ref (list 1 2 3 4) 2) --> 3
```

More service procedures

```
(define (null? x) (equal? x empty))

; takes a list and returns the number
; of elements in the list
(define (our-length list)
  (if (null? list)
      0
      (+ 1 (our-length (cdr list)))))
```

our-member

```
; takes an element and a list and returns non-#f if
; ele is in the list
(define (our-member ele lst)
  (cond ((null? lst) #f)
        ((equal? ele (car lst)) lst)
        (else (our-member ele (cdr lst)))))
```

- (first-n lst n)

Append

```
; takes two lists and returns a list
; containing the elements of the
; original 2
(define (our-append list1 list2)
  (if (null? list1)
      list2
      (cons (car list1)
            (our-append (cdr list1)
                        list2))))

(our-append (list 1 2) (list 3 4))
--> (1 2 3 4)
```

Notes

- Procedures `list-ref`, `null?`, `eq?`, `length`, **and** `append` **are predefined procedures in Scheme**
- **Procedure** `append` **can append any number of lists together**
- **Procedure** `pair?` **returns** `#t` **if its argument is a pair, else** `#f`