#### Next Exam

- Two potential dates
  - $\rightarrow$  Monday, March 18, 2002 or Wednesday, March 20, 2002
- Two hour exam

#### Sections in the book

- Family trees were 14-16
- Multiple complex arguments was 17

## Happy "Thinking Day"

- Shared birthday of Lord & Lady Baden Powell
- Day for reflection on the true meaning of "scouting"

COMP 210, Spring 2002

# Programs with Multiple Complex Arguments

So far, three cases

- Two arguments, one is not inspected
  - $\rightarrow$  Use template for the inspected argument
- Two arguments, with simplifying property
  - $\rightarrow$  Lists of same length
  - $\rightarrow\,$  Trees of identical shape
  - $\rightarrow\,$  Use one argument to control the flow of the program
- Two arguments, no simplifying assumptions
  - $\rightarrow$  Build a table of the cases
  - $\rightarrow$  Develop tests for each case
  - $\rightarrow\,$  Use a cond with a clause for each case
  - $\rightarrow$  Lots of opportunities to recur



1



Example: append

Example: make-points



(in the evening)

Example: merge

Programs with Multiple Complex Arguments

#### Sorting with merge

(list c1 c2 c3 c4 c5 c6 c7 c8)  $\Rightarrow$ 

(list c1) (list c2) (list c3) (list c4) (list c5) (list c6) (list c7) (list c8)  $\downarrow$  merge  $\downarrow$  merge  $\downarrow$  merge (list ci cj) (list ck cl) (list cm cn) (list co cp)  $\downarrow$  merge (list ci cj ck cl) (list cm cn co cp)  $\downarrow$  merge (list ci cj ck cl cm cn co cp)

Question becomes, can we generate singleton lists from a list?

- We do not yet have the tools to do this
- Next section of 210 examines a paradigm that can do this

COMP 210, Spring 2002

## Moving on ...

COMP 210 has mid-term grades due next week ...

- Need software to help compute grades
- Assume that I have a list of scores

;; a list-of-numbers is either

- ;; empty, or
- ;; (cons f r), where f is a number and r is a list-of-numbers
- ;; We will use Scheme's built-in list constructor for list-of-numbers





# Example

(list 72 84 99 53 88 75 104 62)

## Work from standard list template

```
(define (f a-los ...)
(cond
[(empty? a-los) ...]
[(cons? a-los)
... (first a-los) ...
... (f (rest a-los) ... ) ...]
))
```

COMP 210, Spring 2002

## Best-score

#### Filling in the template

;; best-score: list-of-number -> number ;; Purpose: return the best score in the list (define (best-score a-los ...) (cond [(empty? a-los) [(cons? a-los) ... (first a-los) ... ... (best-score (rest a-los) ... ) ... ] ))

### Deep philosophical question

- What is (best-score empty)?
  - $\rightarrow\,$  Since it's a test, we have a lower bound of zero
  - $\rightarrow$  Can return zero



```
5
```

# Best-score

Filling in the template

```
;; best-score: list-of-score -> number
(define (best-score a-los)
  (max-of-list a-los 0))
;; bigger: number number -> number
(define (bigger n1 n2)
 (cond [(<= n1 n2)
                            n2]
                                                            Helper functions
        [else
                            n1] ))
                                                            to make it clean
;; max-of-list: list number -> number
(define (max-of-list a-list lb)
  (cond [(empty? a-list)
                                     lb ]
         [(cons? a-list) (bigger (first a-los) (max-of-list (rest a-los) lb) ]
   ))
```

```
COMP 210, Spring 2002
```

# The Real Problem

The lower bound let us sidestep the issue

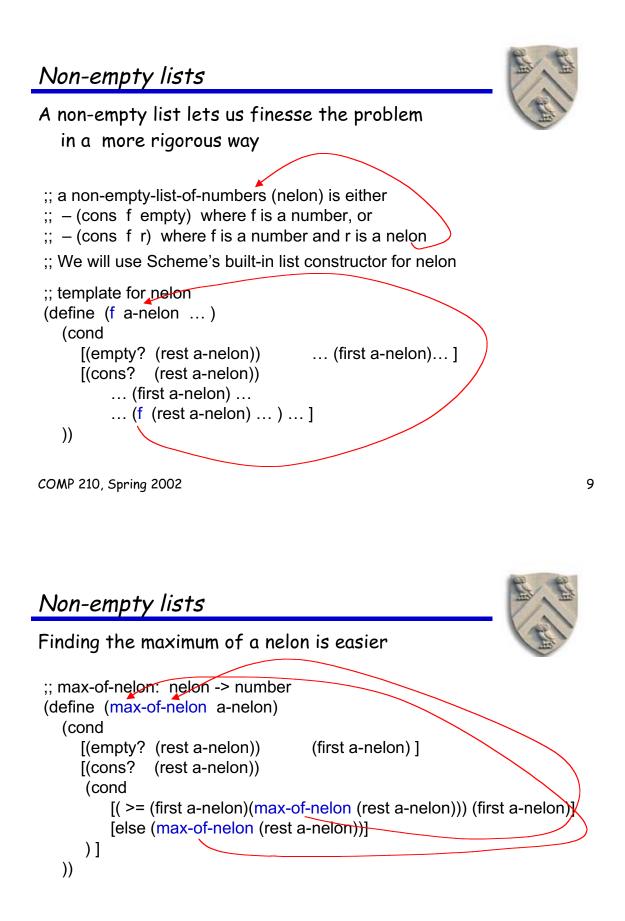
What if we do not have a lower bound?

- (max empty) must return a number
- There is no good answer for this one
  - $\rightarrow$   $\infty~$  is smaller than any other number
  - $\rightarrow$  How can a program that uses max tell if the list actually contained  $\infty$  or not?

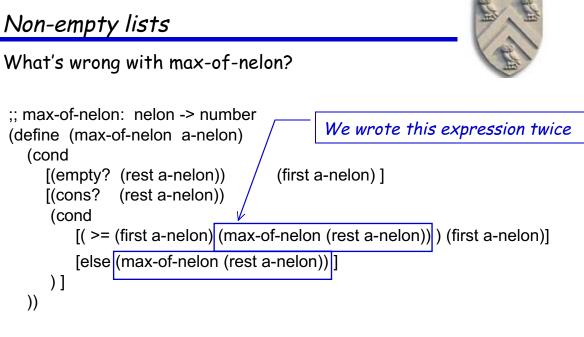
We need another answer to this quandry







We restricted the <u>domain</u> of the inputs to avoid the tricky case - an old and time-honored trick!





COMP 210, Spring 2002

Non-empty lists How bad can it get? Let's try it (max (list 1 2 3 4 5 6)) 1  $\rightarrow$  Recurs twice on (list 2 3 4 5 6) 2 4  $\rightarrow$  Each of those recurs twice on (list 3 4 5 6)  $\rightarrow$  Each of those recurs twice on (list 4 5 6) 8  $\rightarrow$  Each of those recurs twice on (list 5 6) 16  $\rightarrow$  Each of those recurs twice on (list 6) 32  $\rightarrow$  Phew! This is getting ridiculous  $\Rightarrow 63$ It's a little better if the list is not in order, but ...  $\rightarrow$  List of length n calls max 2<sup>n</sup> - 1 times

- $\rightarrow$  This is too much
- $\rightarrow$  List of length 7 would take 127 calls, 8 would take 255, ...





# Need a new (for COMP 210) idea

What's the answer?

- Save the value of <u>max-of-list</u>
- Makes it recur only once
  (max (list 123456))
  - $\rightarrow$  Recurs once on (list 2 3 4 5 6)
    - $\rightarrow$  Recurs once on (list 3 4 5 6)
    - $\rightarrow$  Recurs once on (list 4 5 6)
    - $\rightarrow$  Recurs once on (list 5 6)
    - $\rightarrow$  Recurs once on (list 6)
    - $\rightarrow$  And is done
- Reduces work to n calls for list of length n
  - $\rightarrow$  Exponential savings in work are always worth pursuing

COMP 210, Spring 2002

# Next class

We will introduce a new piece of Scheme syntax

- It will let us save results of temporary computations
- It will improve the power and efficiency of our programs
- It will introduce a critical concept in Computer Science
  - $\rightarrow$  Lexical scoping

# READ INTERMEZZO THREE FOR MONDAY



1

1

1

1

1

1

 $\Rightarrow 6$ 

