

COMP 210, FALL 2000

Lecture 12: Moving Beyond Lists

Reminders:

- Homework assignment due **Friday 2/18/00**
- Exam will be 2/16/2000, in class—closed-notes, closed-book

Review

1. Introduced non-list information structures with the example of a child-centric family tree—that is, a family tree structured from the child's point of view.
2. Build a program **in-family?** that checked a symbol for membership in a family tree. See the posted lecture notes for a correction to what I said about the need for a helper function in **in-family?**.
3. Expanded the utility of family trees in two ways—adding more fields and allowing flexibility to accommodate unknown information with **empty**

Discussion of Exam

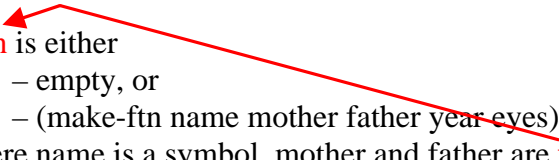
We spent fifteen minutes answering questions about the exam, about templates, and about Scheme issues.

Discussion of Newsgroup Remarks

We spent about five minutes talking about a complaint that I received regarding remarks made on the newsgroup.

Defining a Family Tree, Take 2

```
;; a ftn is either
;;   – empty, or
;;   – (make-ftn name mother father year eyes)
;; where name is a symbol, mother and father are ftn, year is a number,
;; and eyes is a symbol
(define-struct ftn (name mother father year eyes) )
```



```
;; Examples
empty
(make-ftn
  'Mary
  (make-ftn 'Ann empty empty 1950 'blue)
  empty
  1975
  'green )
```

What does the template for this more complex **ftn** look like?

```
(define (f ... a-ftn ... )
  (cond
    [(empty? a-ftn) ... ]
    [(ftn? a-ftn) ...
     (ftn-name a-ftn) ...
     (f (ftn-mother a-ftn) ... ) ...
     (f (ftn-father a-ftn) ... ) ...
     (ftn-year a-ftn) ...
     (ftn-eyes a-ftn) ...
    ]
  ))
```

What does the program **in-family?** look like on this new version of **ftn**?

```
;; in-family?: ftn symbol -> boolean
;; Purpose: returns true if symbol is in the family tree
(define (in-family? a-ftn name )
  (cond
    [(empty? a-ftn) false ]
    [(ftn? a-ftn)
     (or
      (compare-names (ftn-name a-ftn) name )
      (in-family? (ftn-mother a-ftn) name)
      (in-family? (ftn-father a-ftn) name) )
     ]
  ))
```

Develop the program **count-female-ancestors**: ftn -> number. It should return the number of female ancestors in the **ftn**; a person does not count as their own ancestor.

```
;; count-female-ancestors: ftn -> num
;; Purpose: consumes a ftn and returns the number of female ancestors
(define (count-female-ancestors a-ftn)
  (cond
    [(empty? a-ftn) 0]
    [else
     (cond
      [(empty? (ftn-mother a-ftn) (count-female-ancestors (ftn-father a-ftn)))]
      [else (+ 1
              (count-female-ancestors (ftn-mother a-ftn))
              (count-female-ancestors (ftn-father a-ftn)) )]
      )
     ]
  ))
```

This is an edited version of what one student did at the board.