COMP 210, Spring 2001
Lecture 9: Moving Beyond Lists

Reminders:

- New homework assignment due next **Wednesday** 2/13/00
- Exam will be 2/23/2000, in class–closed-notes, closed-book

Review

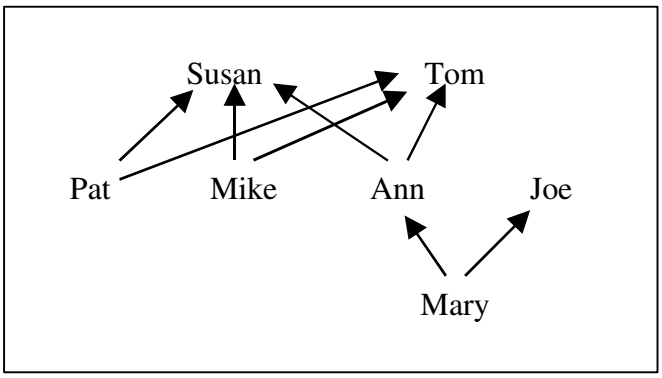
⇒ Talked about programs that deal with the natural numbers.

Design Methodology (Review)

Six steps in the methodology. See page 128 in the book.

Working with Mixed Data

By now you should be comfortable working with lists and with recursion. This gives us the foundation we need to start designing programs that operate over more complex data structures. Today, we'll start by working with family trees.



This family tree depicts three generations of a family. Arrows run from child to parent, so Mary's parents are Ann and Joe, Ann's parents are Susan and Tom, and Pat and Mike are Ann's siblings.

How might we write a data definition that allows us to represent these family trees in Scheme? (Recall that we used a list to represent recipes.) This is where I think Computer Science gets fun–-devising new and effective ways to represent complex kinds of information.

```
;; an FT (for family-tree) is either
;; - a symbol, or
;; - (make-FT n m f)
;; where n is a symbol and f & m are both FT's
(define-struct FT (name mother father))

;; Examples
'Mary
(make-FT 'Ann 'Susan 'Tom)
(make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe)
(make-FT 'Pat 'Susan 'Tom)
(make-FT 'Mike 'Susan 'Tom)
```

Designing Programs for FTs

What would the generic template for this **FT** contain?

```
(define  (f … a-ft …)
  (cond [(symbol? a-ft) … ]
        [(FT? a-ft)
          … (FT-name a-ft)
          … (f … (FT-mother a-ft) …)
          … (f … (FT-father a-ft) …) … ]))
```

Let's write a program **in-family?** that consumes an **FT** and a **symbol** and produces a boolean that indicates whether or not a person with that name is in the family tree.

```
;; in-family?: FT symbol → boolean
;; Purpose: determines if the symbol a-person is in the
;;          FT a-ft
;; (define (in-family? a-ft a-person) …)
```

Next, we can copy the template over and fill it in.

```
(define (in-family? a-ft a-person)
  (cond
    [(symbol? a-ft) (symbol=? a-ft a-person)]
    [(FT? a-ft)
        (or
          (symbol=? (FT-name  a-ft) a-person)
          (in-family? (FT-mother a-ft) a-person )
          (in-family? (FT-father a-ft) a-person))]))
```

We can use **or** to check all three possibilities in a single function call, producing the boolean **or** of the answers.

Should we consider writing a helper function to compare the names? After all, the function occurs in multiple places. The function would look something like

```
(define (compare-names n1 n2)
  (symbol=? n1 n2))
```

This function looks a little ridiculous. It simply passes **n1** and **n2** on to the built-in function **symbol=?** and returns the result. Why would we build a helper function for that?

Well, with **name** implemented as a symbol, writing **compare-names** will make little sense. If, however, names were, themselves, compound objects where the equality test required use of selector functions, or application of multiple equality tests, then abstracting out this function into a helper like **compare-names** would make sense.

Sometimes, you can see these coming. More often, you will discover the need for a helper function like **compare-names** as you are writing the code that needs help. You should still go ahead, create the helper function, and use it to simplify the code. Using a helper function to replace short but complex sequences of code that are repeated makes the resulting code easier to read. It also centralizes the knowledge and control into the helper function–-in the sense that a later change can be made in one place, rather than in many places. This should, in principle, lead to software that is easier to understand, to modify, and to maintain.

If all of the tests on a two-digit year had been isolated into a single helper function, or even a couple (for **=<** and **>**), the Y2K problem would have been much easier to fix.

To finish up with **in-family?** on this version of **family trees**, let's apply the program to some of our example data.

```
    (in-family? 'Joe 'Keith)
⇒   (cond
      [(symbol? 'Joe)  (symbol=? 'Joe 'Keith)]
      [(FT? 'Joe)
        (or (symbol=? (FT-name 'Joe) 'Keith)
            (in-family? (FT-mother 'Joe) 'Keith)
            (in-family? (FT-father 'Joe) 'Keith))]))

⇒   (cond
      [true  (symbol=? 'Joe 'Keith)]
      [(FT? 'Joe)
        (or (symbol=? (FT-name 'Joe) 'Keith)
            (in-family? (FT-mother 'Joe) 'Keith)
            (in-family? (FT-father 'Joe) 'Keith))]))

⇒    (symbol=? 'Joe 'Keith)

⇒    false
```

What about a more complex example?

```
  (in-family?
    (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe)
    'Keith)
⇒ (cond
    [(symbol? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
     (symbol=? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
               'Keith)]
    [(FT? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
     (or (symbol=?
           (FT-name (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith)
         (in-family?
           (FT-mother (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith)
         (in-family?
           (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith))])
⇒ (cond
    [false
     (symbol=? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
               'Keith)]
    [(FT? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
     (or (symbol=?
           (FT-name (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom 'Joe))
           'Keith)
         (in-family?
           (FT-mother (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith)
         (in-family?
           (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom 'Joe))
           'Keith))])
⇒ (cond
    [(FT? (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
     (or (symbol=?
           (FT-name (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith)
         (in-family?
           (FT-mother (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith)
         (in-family?
           (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
           'Keith))])
…⇒… (or (symbol=?
          (FT-name (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
          'Keith)
        (in-family?
          (FT-mother (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
          'Keith)
        (in-family?
          (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))
```

```
…⇒… (or
       (in-family?
         (FT-mother (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith)
       (in-family?
         (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))])
…⇒… (or
       (in-family? (make-FT 'Ann 'Susan 'Tom) 'Keith)
       (in-family?
         (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))

⇒ (or
     (cond
       [(symbol? (make-FT 'Ann 'Susan 'Tom))
        (symbol=? (make-FT 'Ann 'Susan 'Tom) 'Keith)]
       [(FT? (make-FT 'Ann 'Susan 'Tom))
        (or
          (symbol=? (FT-name (make-FT 'Ann 'Susan 'Tom)) 'Keith)
          (in-family? (FT-mother (make-FT 'Ann 'Susan 'Tom)) 'Keith)
          (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom)) 'Keith))])
     (in-family?
       (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
       'Keith))


…⇒… (or
       (cond
         [(FT? (make-FT 'Ann 'Susan 'Tom))
          (or
            (symbol=? (FT-name (make-FT 'Ann 'Susan 'Tom)) 'Keith)
            (in-family? (FT-mother (make-FT 'Ann 'Susan 'Tom)) 'Keith)
            (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom))
                        'Keith))])
       (in-family?
         (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))


…⇒… (or
       (or
         (symbol=? (FT-name (make-FT 'Ann 'Susan 'Tom)) 'Keith)
         (in-family? (FT-mother (make-FT 'Ann 'Susan 'Tom)) 'Keith)
         (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom)) 'Keith))
       (in-family?
         (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))

…⇒… (or
       (or
         (in-family? (FT-mother (make-FT 'Ann 'Susan 'Tom)) 'Keith)
         (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom)) 'Keith))
       (in-family?
         (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
         'Keith))

⇒ (or
```

```
      (or
        (in-family? 'Susan 'Keith)
        (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom)) 'Keith))
      (in-family?
        (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
        'Keith))

…⇒… (or
        (in-family? (FT-father (make-FT 'Ann 'Susan 'Tom)) 'Keith))])
        (in-family?
          (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
          'Keith))

⇒ (or
    (in-family? 'Tom 'Keith)
    (in-family?
      (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
      'Keith))
…⇒… (in-family?
        (FT-father (make-FT 'Mary (make-FT 'Ann 'Susan 'Tom) 'Joe))
        'Keith))
…⇒… (in-family? 'Joe 'Keith)
…⇒… false
```