Administrative Notes

Homework

- Next homework out this afternoon
- Full-length assignment

Lab lecture

- Focus on programs with multiple complex arguments
- This stuff is <u>important</u>
  - → many real programs fit this mold

Programs with Multiple Complex Arguments

So far, three cases

- Two arguments, one is not inspected
  - → Use template for the inspected argument

  Example: **append**
- Two arguments, with simplifying property
  - → Lists of same length
  - → Trees of identical shape

  Example: **make-points**
  - → Use one argument to control the flow of the program
- Two arguments, no simplifying assumptions
  - → Build a table of the cases
  - → Develop tests for each case

  Example: **merge**
  - → Use a **cond** with a clause for each case
  - → Lots of opportunities to recur

Programs with Multiple Complex Arguments

Another example

```
;; merge : list-of-numbers  list-of-numbers -> list-of-numbers
;; Purpose: consumes two lists of numbers, assumed to be in
;;          ascending order by value, and produces a single list of
;;          numbers that contains all the elements of the input lists
;;          (including duplicates) in ascending order by value
(define (merge  a-lon1  a-lon2) …)
```

- Merge must look inside both lists
- The lists can have different length

  → (merge empty (cons 1 empty)) should be (cons 1 empty)

Programs with Multiple Complex Arguments

Merge

- Questions for list x list

	(empty? a-lon2)	(cons? a-lon2)
(empty? a-lon1)	(and (empty? a-lon1) (empty? a-lon2))	(and (empty? a-lon1) (cons? a-lon2))
(cons? a-lon1)	(and (cons? a-lon1) (empty? a-lon2))	(and (cons? a-lon1) (cons? a-lon2))

The template must include (and handle) all these cases

Programs with Multiple Complex Arguments

Merge – the template

```
(define (f  a-lon1  a-lon2)
  (cond
   [(and (empty? a-lon1) (empty? a-lon2))     …]
   [(and (empty? a-lon1) (cons? a-lon2))
          … (first a-lon2) …  (f  a-lon1 (rest a-lon2)) …]
   [(and (cons? a-lon1) (empty? a-lon2))
          … (first a-lon1) …  (f (rest a-lon1) a-lon2)…]
   [(and (cons? a-lon1) (cons? a-lon2))
          … (first a-lon1) … (first a-lon2) …
          … (f  a-lon1 (rest a-lon2)) …
          … (f (rest a-lon1) a-lon2) …
          … (f (rest a-lon1) (rest a-lon2)) …]
  )
)
```

May not need all the possible recursions

Programs with Multiple Complex Arguments

Merge – the program

```
(define (merge  a-lon1  a-lon2)
  (cond
   [(and (empty? a-lon1) (empty? a-lon2))     empty]
   [(and (empty? a-lon1) (cons? a-lon2))      a-lon2]
   [(and (cons? a-lon1) (empty? a-lon2))      a-lon1]
   [(and (cons? a-lon1) (cons? a-lon2))
    (cond
      [(< (first a-lon1) (first a-lon2)
       (cons (first a-lon1) (merge (rest a-lon1) a-lon2))]
      [else
       (cons (first a-lon2) (merge a-lon1 (rest a-lon2)))]
    )
   ]
  )
)
```

Programs with Multiple Complex Arguments

What good is **merge**?

- Forms the core of a general algorithm for sorting
- To sort a list
  - → Break list into lists of length one (or two)
  - → Merge adjacent lists, merge results, …

The result is method of choice for sorting sets of data that are too large to fit in memory

Programs with Multiple Complex Arguments

Sorting with **merge**

(list c1 c2 c3 c4 c5 c6 c7 c8) ⇒

(list c1) (list c2) (list c3) (list c4) (list c5) (list c6) (list c7) (list c8)

⇓ merge ⇓ merge ⇓ merge ⇓ merge

(list ci cj) (list ck cl) (list cm cn) (list co cp)

⇓ merge ⇓ merge

(list ci cj ck cl) (list cm cn co cp)

⇓ 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

Methodology

Wrapping up this set of ideas

Philosophy

- In general, there is only one template for a pair of arguments
- For list x list, it's the full template we developed for **merge**
- We may, however, simplify that template
  - → Problem-specific knowledge, as in **append** or **make-points**
  - → These simplified templates speed up development
  - → These simplified templates may lead to cleaner programs

Programs with Multiple Complex Arguments

Append

- Used the form of the standard list template

```
(define (f  list1  list2 )
   (cond
      [(empty? list1)          … ]
      [(cons?   list1)
            … (first list1) …
            … (f (rest list1) list2 ) …]
   )
)
```

Normally, other arguments are ellided

- Key was to recognize that list2 is uninspected

Programs with Multiple Complex Arguments

Make-points

- Recognized that both lists must have same length

```
(define  (f  x-list  y-list … )
   (cond
     [(empty?  x-list)                    …]
     [(cons?    x-list)
           … (first x-list) … (first y-list) …
           … (f  (rest x-list)  (rest y-list) … ) … ]
   )
)
```

Only need to test one argument

- Simplifies the **cond** structure

Programs with Multiple Complex Arguments

Merge

- Needed the full template for two lists

```
(define (f  a-lon1  a-lon2)
  (cond
    [(and (empty? a-lon1) (empty? a-lon2))     …]
    [(and (empty? a-lon1) (cons? a-lon2))
          … (first a-lon2) …  (f  a-lon1 (rest a-lon2)) …]
    [(and (cons? a-lon1) (empty? a-lon2))
          … (first a-lon1) …  (f (rest a-lon1) a-lon2)…]
    [(and (cons? a-lon1) (cons? a-lon2))
          … (first a-lon1) … (first a-lon2) …
          … (f  a-lon1 (rest a-lon2)) …
          … (f (rest a-lon1) a-lon2) …
          … (f (rest a-lon1) (rest a-lon2)) …]
  )
)
```

Others are special cases of this one

Programs with Multiple Complex Arguments

Philosophy

- In general, there is only one template for a pair of arguments
- For list x list, it's the full template we developed for **merge**
- We may, however, simplify that template
  - → Problem-specific knowledge, as in **append** or **make-points**
  - → These simplified templates speed up development
  - → These simplified templates may lead to cleaner programs

But,

- They are still special cases of the general template