MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Recitation 7, March 1

Higher Order Procedure Problems

Dr. Kimberle Koile

The first idea:

Write a procedure that sums all the integers between (and including) two integers.
Write a procedure that sums the squares of intervening integers.
Write a procedure that sums the factorials of intervening integers.
Notice the similarities in the procedure definitions, and write a general sum procedure that takes an additional argument that is a function to be called on each intervening integer.
Conclusion: One can write a general higher-order procedure for summing between two integers.

The second idea:

Write a procedure that sums the factorial of every other integer between (and including) two integers.
Write a more general sum procedure that takes both a function to call on the integers and a function for finding the next intervening integer.
Conclusion: One can write a more general higher-order procedure that abstracts two functions:
the value-to-be-summed generator and the next integer generator.

The third idea:

Write a procedure similar to the general sum procedure (that takes two integers and two functions as arguments) but that multiplies rather than sums values.
Write an even more general procedure that abstracts three functions: the way to combine values, the way to generate values to be combined, the way to get the next integer for producing a value.
Conclusion: …

The advanced question: Why does calling the even more general procedure with division instead of multiplication or addition) not give the expected answer?

First idea:

1. Write a recursive procedure that sums all integers between (and including) two integers.
(sum-ints 2 5) => (+ 2 3 4 5) => 14

```
(define (sum-ints a b)
  (if (> a b)
      0
      (+ a (sum-ints (+ a 1) b))))
```

2. Write a recursive procedure that sums the squares of all integers between (and including) two integers.
(sum-sq 2 5) => (+ 4 9 16 25) => 54

```
(define (sum-sq a b)
  (if (> a b)
      0
      (+ (* a a) (sum-sq (+ a 1) b))))
```

3. Write a rccursive procedure that sums the factorials of all integers between (and including) two integers.

```
(define (sum-facts a b)
  (if (> a b)
      0
      (+ (fact a) (sum-facts (+ a 1) b))))
```

4. Write a recursive procedure that sums the results of calling a specified function on all integers between (and including) two integers. (Hint: Pick out the common pattern in your answers to problems 2 and 3.)

```
(define (sum-genl a b fn)
   (if (> a b)
      0
      (+ (fn a) (sum-genl (+ a 1) b fn))))
```

5. Write sum-ints to use sum-genl.

```
(define (sum-ints  a  b)
   (sum-genl a b (lambda(x) x))
```

Second idea:

6. Write a procedure to sum the factorial for every other integer between (and including) two integers.

```
(define (sum-every-other-fact a b)
   (if (> a b)
       0
       (+ (fact a)(sum-every-other-fact (+ a 2) b))))
```

7. Write a general sum procedure that takes a function to be called on each integer to be summed and a function for finding the next integer to sum.

```
(define (sum-more-genl a b fn next)
  (if (> a b)
      0
     (+ (fn a)(sum-more-genl (next a) b fn next))))
```

Third idea:

8. Write a procedure similar to sum-more-genl, but that multiplies the results of calling a function on integers between two integers, rather than adding them.

```
(define (prod a b fn next)
  (if (> a b)
      1    ;;; not a 0!
     (* (fn a)(prod (next a) b fn next))))
```

9. Write an even more general procedure that abstracts three functions: the way to combine values, the way to generate values to be combined, the way to get the next integer for producing a value. (Hint: It takes six arguments.)

```
(define (general-combine combiner a b fn next ident)
   (if (> a b)
      ident
     (combiner (fn a)(general-combine combiner (next a) b fn next ident))))
```

10. What happens when we call the procedure in question 9 with / (division)? Why don't we get the answer that we expect? Work out an example with the substitution model and see.

We do not get the answer that we expect because division is not commutative. Conceptually, we expect the combiner, for example, to sum ints from 1 to 3 as 1 + 2 + 3. Because of the recursion, however, the addition actually happens in the opposite order, starting with 3: (+ 1 (+ 2 (+ 3 0))). With division, we expect (/ (/ (/ 1 2) 3) 4) but instead get (/ 2 (/ 3 (/ 4 1))).