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

Recitation 5, Friday, February 23

Recursion Review Dr. Kimberle Koile

What is the difference in calling pattern between recursive factorial, iterative recursive factorial, and tail recursive iterative factorial? (By "calling pattern" I mean how procedures call other procedures and return values.)

Recursive factorial

```
(define (fact  n)
   (if (= n 1) 1 (* n (fact (- n 1)))))
```

```
 1. (fact 4)
 2. (* 4 (fact 3))
 3. (* 4 (* 3 (fact 2)))
 4. (* 4 (* 3 (* 2 (fact 1))))
 5. (fact 1) returns 1, which is substituted into pending (* 2 <result>)
 6. pending (* 2 1) returns the value 2 to its caller, (fact 2), in step 3
 7. (fact 2) returns the 2
 8. pending (* 3 2) returns the value 6 to its caller, (fact 3), in step 2
 9. (fact 3) returns the value 6
10. (pending (* 4 6) returns the value 24 to its caller, (fact 4), in step 1
11. (fact 4) returns the value 24
```

Iterative recursive factorial

```
(define ifact (lambda (n) (ifact-helper 1 1 n)))

(define helper (lambda (product counter n)
     (if (> counter n)
      product
      (helper (* product counter) (+ counter 1) n))))
```

```
 1. (ifact 4)
 2. (helper 1 1 4)
 3. (helper 1 2 4)
 4. (helper 2 3 4)
 5. (helper 6 4 4)
 6. (helper 24 5 4)
 7. (helper 24 5 4) returns 24 to caller in step 5, which was (helper 6 4 4)
 8. (helper 6 4 4) returns the 24 to caller in step 4
 9. (helper 2 3 4) returns the 24 to caller in step 3
10. (helper 1 2 4) returns the 24 to caller in step 2
11. (helper 1 1 4) returns the 24 to caller in step 1
12. (ifact 4) returns 24
```

Iterative tail recursive factorial

Notice that with iterative recursive factorial, no operations were performed on the value 24 after step 7; the value was simply passed back to the calling procedures.

This situation---when the recursive call itself is the last operation in a procedure, is called *tail recursion*. A compiler can identify such situations and optimize the code to return the value after the innermost recursive call completes.

With the optimization for tail recursion:

```
1. (ifact 4)
2. (helper 1 1 4)
3. (helper 1 2 4)
4. (helper 2 3 4)
5. (helper 6 4 4)
6. (helper 24 5 4)
7. (helper 24 5 4) returns 24 to outermost caller, (ifact 4)
8. (ifact 4) returns 24
```