Creating Functional Programs

Brad Vander Zanden

Basic Techniques

- Tail Recursion
  - Use continuation arguments if necessary
  - Akin to pre-processing a list
- Inductive Construction
  - Akin to post-processing a list

Factorial

- Inductive Construction

```
(define fact (lambda (n)
   (cond
    ((= n 0) 1)
    ((= n 1) 1)
    (#t (* n (fact (- n 1)))))))
```

Factorial

- Tail Recursion Construction

```
(define fact (lambda (n)
   (letrec ((factHelper (lambda (n productThusFar)
     (cond
         ((= n 0) productThusFar)
         ((= n 1) productThusFar)
         (#t (factHelper (- n 1) (* n productThusFar)))))))
     (factHelper n 1))))
```

Sum of Numbers

- Inductive Construction

```
(define sumList (lambda (L)
    (cond
     ((null? L) 0)
     (#t (+ (car L) (sumList (cdr L)))))))
```

Sum of Numbers

- Tail Recursion Construction

```
(define sumList (lambda (L)
   (letrec ((sumListHelper (lambda (L sumThusFar)
                (cond
                    ((null? L) sumThusFar)
                    (#t (sumListHelper (cdr L)
                                        (+ (car L) sumThusFar)))))))
    (sumListHelper L 0))))
```

Quicksort

- Inductive Construction

```
(define qsortPartition (lambda (pivot L)
 (if (null? L)
   (cons '() '())
   (let* ((result (qsortPartition pivot (cdr L)))
          (lesserList (car result))
          (greaterList (cdr result)))
       (if (< (car L) pivot)
          ; add the head of L to the lesserList
         (cons (cons (car L) lesserList) greaterList)
          ; add the head of L to the greaterList
         (cons lesserList (cons (car L) greaterList)))))))
```

Quicksort

- Tail Recursion Construction

```
(define qsortPartition (lambda (pivot L L1 L2)
  (if (null? L)
     (cons L1 L2)  ; return L1 and L2 once L is exhausted
     (let ((firstElement (car L)))
           (if (< firstElement pivot)
              ; add head of L to L1 and partition rest of L
              (qsortPartition pivot (cdr L)
                                         (cons firstElement L1) L2)
              ; add head of L to L2 and partition rest of L
              (qsortPartition pivot (cdr L) L1
                                            (cons firstElement L2)))))))
```

Quicksort

- Either construction

```
(define qsort (lambda (L)
  (if (null? L)
     L
     (let* ((result (qsortPartition (car L) (cdr L) '() '()))
             (lesserList (car result))
             (greaterList (cdr result)))
      (append (qsort lesserList)
                 (list (car L))
                 (qsort greaterList))))))
```