CS 135 Fall 2015

Tutorial 6: Return of the Midterm

Reminder: Monday November 2nd

- The midterm will be held on Monday November 2nd at 7:00PM.
- Seating for the midterm will be posted on the course webpage on Friday morning.
- The midterm will cover up to and including module 07.
- There will be **NO** assignment due Tuesday November 3rd.

Midterm Practice

- Redo assignments
- Complete tutorial problems
- Complete the warmup and practise problems listed on assignments
- Make up practice questions for yourself
- Reread the course notes
- Every question in this tutorial was inspired by a point in a module summary

Clicker Question - List Translation

Given this list:

```
(list (list) 'cons (list (list 2 'green) 3))
```

Which of the following is equivalent to the given list?

A '(empty cons (list 2 'green) 3)

B '(empty cons (2 'green) 3)

C '(empty cons (list (list 2 'green) 3))

D '(() cons ((2 green) 3))

E '(() 'cons ((2 'green) 3))

Group Problem - Steppers

Give the first and second substitution steps as well as the final value for the following expression.

```
(length (rest (rest (second '((hello red) (0 1 1 2 3 5) ())))))
```

Structural and Structural w/Accumulator

In **(pure) structural recursion**, all parameters to the recursive call(s) are either unchanged, or *one step* closer to a base case according to the data definition. All parameters must be either ride-along or one-step parameters.

In **structural recursion with an accumulator**, parameters meet the requirements of (pure) structural recursion, plus one or more accumulators, or parameters containing partial answers. To identify this type of recursion, ususally you can check to see if the base case returns an accumulator.

Generative Recursion

In **generative recursion**, at least one parameter which is being recursed on is being modified in a way not in accordance with its data definition (watch out for correctness and termination). A simple example is given below:

```
;; A Nat is one of:
;; * 0
;; * (add1 Nat)
```

Generative Recursion

```
(define (func x)
  (cond [(zero? x) (cons 0 empty)]
        [else (cons x (func (− x 2)))]))
```

This function does not follow the data definition for natural numbers. The parameter x goes two steps closer to the base case in the recursive application. If x is given an odd number the function will not terminate.

Clicker Question - Types of Recursion

```
(define (bizz x y)
  (cond
    [(empty? x) y]
    [(number? (first x)) (bizz (rest x) (+ (first x) y))]
    [(string? (first x))  (bizz (rest x) (+ y (string-length (first x))))]
    [else (bizz (rest x) y)]))
```

What type of recursion is this?

A Structural
B Structural w/Accumulator
C Generative

Clicker Question - Types of Recursion

```
(define (fizz n m)
  (cond
    [(<= (/ n 2) m) empty]
    [(integer? (/ n m)) (cons m (cons (/ n m) (fizz n (add1 m))))]
    [else (fizz n (add1 m))]))
```

What type of recursion is this?

A Structural
B Structural w/Accumulator
C Generative

Clicker Question - Types of Recursion

```
(define (boom x y z)
  (cond [(empty? x) empty]
        [(and (equal? (first x) y) (not (empty? (rest x))))
         (boom (rest (rest x)) y (min z (length x)))]
        [(and (number? (first x)) (> z (first x)))
         (cons (first x) (boom (rest x) y (max z (length x))))]
        [else (boom (rest x) y (max z (length x)))]))
```

What type of recursion is this?

A Structural
B Structural w/Accumulator
C Generative

Group Problem - anagram?

An anagram is a rearrangement (permutation) of the letters of a word. For example, “ate” is an anagram of “tea”, and “foster” is an anagram of “forest”. Write a predicate function called anagram? that determines if two strings are anagrams of each other, producing true if they are. You may assume that the input strings contain only lower-case letters (i.e., no spaces, punctuation, or upper-case letters).

Group Problem - Sub-list

You will write a function called sub-list which has 3 parameters. Two natural numbers denote the range of elements to extract from the list. If the list doesn't have sufficient elements at any point then any contents within the range so far are returned. Note that the first index of a list is 0.

```
;; (sub-list begin end lst) produces elements from index begin, to index end of the list lst.
;; sub-list: Nat Nat (listof Any) ⟶ (listof Any)
;; requires: end >= begin
;; Examples:
(check-expect (sub-list 9 10 '(I knew you were trouble when you walked in trouble trouble trouble))
              '(trouble trouble))
(check-expect (sub-list 3 9 '(a b c d)) '(d))
;; Tests:
(check-expect (sub-list 0 0 '()) '())
(check-expect (sub-list 3 5 '(a b c d e f g h)) '(d e f))
```

Group Problem - palindrome?

A palindrome is a sequence of characters which reads the same backward or forward. You will write a predicate called palindrome?. The function takes a list of characters as a parameter and identifies if the list of characters is a palindrome. Include the contract from below:

```
;; (palindrome? loc) produces true if loc is a palindrome.
;; palindrome?: (listof Char) ⟶ Bool
;; Examples:
(check-expect (palindrome? (list #\p #\u #\t #\i #\t #\u #\p)) true)
(check-expect (palindrome? empty) true)
;; Tests:
(check-expect (palindrome? (list #\n #\u #\m #\b #\e #\r)) false)
(check-expect (palindrome? (list #\e #\l #\l #\e)) true)
(check-expect (palindrome? (list #\g #\l #\u)) false)
```