CS 135 Fall 2018
Midterm 2 Help Session
Reminder: Midterm 2 (October 29)

- The midterm will be held on Monday, October 29 at 7:00 PM.
- Check your seating for the midterm on Odyssey.
- The midterm will cover up to and including the end of Module 06.
- There will be **NO** assignment due Tuesday, October 30.
Clicker Question - box-and-pointer

Which of the following nested box representations match this box-and-pointer representation?

A

B

C

D

E
Clicker Question - List Translation

Given this list:

\[
\text{(list (list) 'cons (list (list 2 'green) 3))}
\]

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))
Stepping through Lists

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

\[
\text{(length (rest (rest (second '((hello red) (0 1 1 2 3 5) ())))))}
\]
Template functions - Shapes

Consider the following definitions:

(define-struct rectangle (length width colour))
;; A Rectangle is a (make-rectangle Num Num Sym)
;; requires: length, width > 0

(define-struct triangle (base height colour))
;; A Triangle is a (make-triangle Num Num Sym)
;; requires: base, height > 0

;; A Shape is (anyof Rectangle Triangle)

Write a template function for a Shape and another template function for a list of Shapes.
Insertion Sort - sort-shapes

Using your template functions as a guide, write a function called sort-shapes that sorts a list of Shapes in non-decreasing order of area. If two shapes have the same area, they should appear in the same order as in the original list. The ordering of a rectangle vs a triangle does not matter.
Recursing on a list and 2 Nats - sublist

Write a function called `sublist` which consumes a list, `lst`, and 2 natural numbers, `start` and `end`. `sublist` should produce the elements in `lst` indexed from `start` up to but not including `end`. 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 element of a list is indexed at 0.

```
(sublist '(a b c d e f) 2 5) ⇒ '(c d e)
(sublist '(a b c d e f) 4 8) ⇒ '(e f)
```
2-dimensional lists - get-table-chunk

Consider the following data definition:

;; A Table is a (listof (listof Any))
;; requires: all the sublists have the same length

Using sublist, write a function called get-table-chunk which consumes a Table and 4 natural numbers, col-start, col-end, row-start and row-end. get-table-chunk should produce the table with only rows from row-start up to but not including row-end with their columns indexed from col-start up to but not including col-end. You may assume the input is valid. Note that columns and rows’ indices start at 0.

(get-table-chunk '((1 2 3 4 5) (a b c d e) (3 6 9 12 15) (f g h i j)) 1 3 1 3)
⇒ '((b c) (6 9))

(get-table-chunk '((1 2 3 4 5 7) (a b c d e 8) (3 6 9 12 15 8) (f g h i j l) (f g h i j l) (f g h i j l)) 1 4 0 3)
⇒ '((2 3 4) (b c d) (6 9 12))
Recursing on a Nat - add

In this problem, we will be implementing the addition of 2 Nats without using the built-in Racket function $+$. Write a function called add that adds together 2 natural numbers. The only built-in arithmetic functions you may use are add1 and sub1. You may not use any helper functions.
Recursing in lockstep - hangman

In the game of hangman, one player decides on a secret word and the other player tries to guess the word one letter at a time. In this problem, we will write a function that simulates one such guess.

Write a function called `hangman` that consumes a string called `secret-word` and another string called `current-state`, as well as a single character `guess`. `current-state` is the same string as `secret-word` except all the letters that have not been guessed yet are replaced by " \_ ". `hangman` should produce a new string such that if `guess` is in `secret-word`, all the corresponding blanks in `current-state` are replaced by `guess`. Otherwise, `current-state` is produced.

```plaintext
(hangman "cat" "c_t" \a) ⇒ "cat"
(hangman "boo" "___" \o) ⇒ "_oo"
(hangman "onion" "_n_n" \p) ⇒ "_n_n"
```
Recursing at different rates - compute-average

;; A Grade-list is one of:
;; * empty
;; * (cons (list Str Num) Grade-list)
;; requires: strings in a grade-list are sorted using string< and are unique
;; numbers in the grade-list are between 0 and 100, inclusive

Write a function compute-average that takes in 2 grade-lists and produces one grade list combining students from both grade-lists sorted by string<. If a student appears in both of the grade-lists, their new grade is the average of their grades from both of their classes.
Recursing at different rates - compute-average
Here are a few examples:

```lisp
(compute-average (list (list "Jason" 95) (list "Jimmy" 69)) (list (list "Anne" 90) (list "Jason" 87)))
⇒ (list (list "Anne" 90) (list "Jason" 91) (list "Jimmy" 69))

(compute-average (list (list "Jason" 100) (list "Jimmy" 69)) (list (list "Jason" 100) (list "Jimmy" 69)))
⇒ (list (list "Jason" 100) (list "Jimmy" 69))

(compute-average (list (list "Jason" 99) (list "Jimmy" 70)) (list (list "Anne" 90) (list "Jason" 100)))
⇒ (list (list "Anne" 90) (list "Jason" 99.5) (list "Jimmy" 70))
```
Mastering the Design Recipe

Following the design recipe helps you understand the problem and produce correct code. Here are the complete steps in tackling a problem using the design recipe:

- Read the question and summarize your task using a purpose.
- Next, determine the types for all of your input and output and express them in the form of a contract.
- Think of some valid input to the problem and calculate the output manually. These are your examples.
- After writing your function, test your function thoroughly by considering edge cases related to your function.
Templates

We write a template for functions that consume compound data. Here are a few things to pay attention to:

- If it is a structure, you need to select all of its fields.

;;; songinfo-template: SongInfo → Any
(define (songinfo-template info)
  (\ldots (songinfo-performer info)\ldots
    (songinfo-title info)\ldots
    (songinfo-genre info)\ldots
    (songinfo-length info)\ldots))
Templates

- If it is a list, you need to consider whether it is an empty list. For a non-empty list, you need to process the first item and the rest of the list.

;; listof-str-template: (listof Str) → Any
(define (listof-str-template los)
  (cond [(empty? los)...]
        [else (... (first los)... (rest los)...)]))

- In general, if your data definition has "one of..", which also includes lists, then include a conditional expression with one test for each possibility.
Recursion Strategies

In this course, we looked at the following ways of recursion:

- recursion on a number, such as adding numbers from 0 to n
- recursion on a list, such as adding numbers in a list
- recursion on a list and a number, such as getting the ith item of a list
- recursion on 2 lists (locked step or different rates), such as taking the dot product or merging 2 lists
- recursion on 2D-list/nested lists

To succeed in recursion questions, identify the type(s) of recursion you need to use and apply them effectively.
Optional - Recursing on a Nat3 - divisible-by-3?

Consider the following data definition for a natural number:

;; A Nat3 is one of
;; * 0
;; * 1
;; * 2
;; * (+ 3 Nat3)

Write a function called `divisible-by-3?` that consumes a Nat3 and produces whether the input number is divisible by 3. Use pure structural recursion.