Topics for Final Exam

You are allowed to bring two 8.5 x 11 crib sheets to the exam.

Functions

- Rewriting standard functions (e.g. map, filter, reduce, etc) over lists or other data structures
- Functions over lists or data structures with ref cells
- Functions over trees and other datatypes: map, filter, mirror, etc
- Higher order functions (e.g. compose)
    - Using composition of higher order functions to define other functions

Using functions to define new types and functions over these types

- For example
    - Church numerals, Church pairs, Church Booleans with operations
    - Complex Numbers with addition, mult, etc

Exceptions

- Defining exceptions and handlers
- Using exceptions as a programming tool for backtracking

Continuations

- Using continuations for backtracking, tail recursion.

Lazy programming

- Generating streams (e.g. (x, f(x), f(f(x)), f (f(f(x))), …)

Proofs by Induction:

- Some examples

```
fun rev [] = [] | rev (h::t) = (rev t) @ [h]
fun rev_tl ([], acc) = acc | rev_tl (h::t, acc) = rev_tl(t, h::acc)


fun map f [] = [] | map f (h::t) = (f h)::(map f t)
fun map_tl f [] acc = acc | map_tl f (h::t) acc = map_tl f t (acc @ [f h])

fun size Empty = 0 | size Node(_, L, R) = 1 + size L + size R
fun mirror Empty = Empty | mirror (Node(v, L, R))= Node(v,mirror R,mirror L)
```

Prove that `size t = size (mirror t).`

- References and ephemeral data structures
    - Defining ephemeral data structures, e.g. queue, deque, tree
    - Defining functions over such structures (e.g. queue operations, tree delete)
- Environment diagrams
- Language definition and extension
    - Given new productions that extend a language:
        - define the free variables
        - define substitution rules
- Type Inference
    - Given expression, determine its principal type and explain how it was derived
    - Explain why a given expression does not type check