CS 320 − 20 October 2016

Topics Mentioned/Discussed So Far

1. **avoid recomputation**, **memoization**/**dynamic programming**

2. **higher-order functions**, **functions as first-class objects**

3. **infinite lists** (defined but not evaluated)

4. **lazy evaluation** (infinite lists evaluated as much as needed)

5. **lazy evaluation**/**call-by-name evaluation**
   versus **eager evaluation**/**strict evaluation**/**call-by-value evaluation**

6. **recursion** more powerful than other mechanisms for repetitive execution

7. **reasoning about infinite lists** −
   what list is defined by "`nats = 0:(map (1+) nats)`"?
   what list is defined by "`twos = 1:(map (2*) twos)`"?
   what list is defined by "`fibs = 0:1:(zipWith (+) fibs (tail fibs))`"?

8. **tail recursion** versus **recursion in general**

9. **tail recursion** via **continuation-passing style** or **accumulating-parameter style**,

10. **type inference**, **type checking**

11. **polymorphic types**, **monomorphic types**

12. **scoping**, **encapsulation**, **implementation hiding**, **help functions**

13. **local definitions** versus **global definitions**

14. **imperative programming**/**programming with side effects**

15. **(pure) functional programming**/**programming without side effects**,

16. **referential transparency**/**functional programming** versus **referential opaqueness**/**imperative programming**

17. **pattern matching**

18. **list comprehension**

19. **vocabulary**/**lexing**/ **parsing**/**tokens**

20. **regular expressions**/**regular grammars** and **regular languages**,
    context-free grammars/**BNF** and **context-free languages**

21. **associativity** and **commutativity** of binary operators

22. **rules of precedence** of binary operators, **implicit parenthesization**

23. **rules of association** of binary operators, **implicit parenthesization**

24. **function application** (in program code) is left-associative: "MNP"
    means "$((MN)P)$"

25. **arrow constructor "→"** (in type expressions) is right-associative:
    "$t_1 \to t_2 \to t_3$" means "$(t_1 \to (t_2 \to t_3))$"

26. **curried**/**consumes arguments sequentially**

versus **uncurried**/**consumes arguments simultaneously**

27. **user-defined types**, **in Haskell with keywords:** **type**, **newtype**, **data**

28. **recursively defined datatypes** (also called **algebraic datatypes**), introduced with keyword **data**

29. **binary trees**, different versions (labels at the internal nodes only, labels at the leaf nodes only, labels at both), polymorphic or monomorphic

30. **constructors** and **selectors** on binary trees (and other recursively defined datatypes)

31. **foldr**, **foldl**, **foldl'** on lists and their generalizations on recursively defined datatypes (e.g., `trees` using keyword **data**)

32. **map**, **filter**, and other list functions and their generalizations on recursively defined datatypes (e.g., `trees` using keyword **data**)

33. **type classes** and **instance of classes**

34. **most general types** (also called **principal types**)