PROGRAMMING IN HASKELL

Road map

Chapter 1: Introduction

- History of Haskell

Next> Chapter 2: The Basics

Chapter 2: The Basics

- Glasgow Haskell Compiler (ghci)
- Maths: 34.5 + (3.2 * 4.5)
- Lists: head [1,2,3]
- Function application: f x y
- Haskell .hs scripts

Next> Chapter 3: Types and Currying

Chapter 3: Types and Currying

- Types
- Lists and tuples
- Function types and Currying; eg Int -> Int -> Int
- Polymorphism: [a] -> a
- Overloading: Num a => [a] -> a

Next> Chapter 4: Defining Functions

Chapter 4: Defining functions

- Conditional expression: if …. then … else
- Guarded expressions (using |)
- Pattern matching
- Lambda expressions: eg (\x -> x+x)
- Operators: (+) 2 3; (+2) 3

Next> Chapter 5:

List Comprehension and Strings

Chapter 5: List Comprehension and Strings

- List comprehension:

```
[sqrt(x^2+y^2) | x<-xs, y<-ys]
```

- List comprehension with guards:

```
[sqrt(x^2+y^2) | x<-xs, y<-ys, x<y]
```

- Strings: String is [Char]: “abc”

Next> Chapter 6: Recursive Functions

Chapter 6: Recursive Functions

- Recursive functions
- Recursion on lists
- Mutual recursion (two functions)
- Computational efficiency (eg tail recursion)

Next> Chapter 6: Higher-order Functions

Chapter 7: Higher-order Functions

- Functions as values: eg v = (\x -> x+x)
- Functions as arguments: eg map f list
- List aggregation: foldr
- Composition: f . g x = f (g x)
- Returning function values

Next> Chapter 8: Defining Types

Chapter 8: Defining Types

- Declaring types: type Board = [Int]
- Data declarations:

  data Answer = Yes | No | Unknown
- Constructors: eg Rectangle Float Float
- Parametric data types: eg data Tree a
- Recursive data types:

  data Nat = Zero | Succ Nat

Next> Chapter 10: Interactive Programming: IO

Chapter 9 does not exist for us!!!

Chapter 10: Interactive programming; IO

- The type IO a
- Sequencing using do block and return
- putStrLn, getLine: writing and reading from the terminal
- Recursion in sequenced code.

Next> Chapter 15: Lazy Evaluation

Chapter 15: Lazy Evaluation

- Evaluation by application of definitions
- Innermost and outermost reduction
- Sharing thunks
- Lazy evaluation = Outermost reduction + Sharing
- Lazy evaluation is efficient strategy
- Infinite lists: ones = 1:ones

End of lecture notes!