Haskell
Lecture 4

I/O

Higher-Order Functions

String functions

- *A String is a list of Char*
- *words breaks a String into a list of Strings (words) separated from each other by white space in original String. unwords is the reverse*
- *lines and unlines is the same as words and unwords except that '\n' is the separator*

I/O

- *I/O operations are not pure, there are side effects, the results of calling an I/O function can be different when called using same arguments.*
- *The results we want are wrapped in an IO type to demonstrate impurity of that function*
- *Example:*
  ```
        getLine :: IO String
  ```
- *getLine performs input and wraps the input line in IO*
- *Unwrap the value using <- as in*
- *line <- getLine*

Main module and main function

- *main has type IO something*
- *() type is called the unit, it indicates that IO wraps no value*
- *putStr :: String -> IO ()*
- *putStr outputs a string and evaluates to a wrapped unit, which has no information*

do Notation

- *Often, main is written as*
  - *main = do*
- *This allows sequencing of commands and assignments*
- *Look at Main.hs, part of current project*
- *Recursion to simulate input loop*
- *System.Environment for command line argument, System.IO to do I/O with files*

Higher-Order Function

- *A function as an argument to another function*
- *A function returning a function as its result*

map

- *map :: (a -> b) -> [a] -> [b]*
- *map takes a function and a list and has the value of the list created by applying the argument function to each element in the argument list*
- *use map and sum to define length*

filter

- *filter (a -> Bool) -> [a] -> [a]*
- *isAlpha (in Data.Char module) gives value True when applied to a Char value that is alphabetic*
- *filter creates a list using just those elements in the argument list for which the function yields value True*
- *to get just the alphabetic characters from a list of Char (a String), use filter with isAlpha as in*
  - *filter isAlpha "Hello, world!"*

folding

- *foldr1 takes a binary function and a list and combines the elements of a list (right most association) to give a single value. This fails on an empty list*
- *foldr1 (-) [1,3,10] gives 1 - (3 - 10) == 8*
- *foldr takes an extra argument which is the initial value of fold. foldr (-) 0 [1,3,10] same result as above*
- *foldl1 and foldl similar, but associate to left*
- *see file folds.hs*

takeWhile and dropWhile

- *takeWhile has Bool function (a -> Bool) and a list argument and creates the list taken from beginning of list argument for which function yields True*
- *dropWhile works the same way, except it removes the beginning values for which function returns true*
- *takeWhile isAlpha "Hello world" gives "Hello"*
- *dropWhile isAlpha "Hello world" gives " world"*

interact

- *Higher-order I/O function*
- *interact :: (String -> String) -> IO ()*
- *Takes a function from String to String, applies input from stdin as argument to function, writes output of function to stdout*
- *Examples: use with lines, unlines, words, unwords*

Converting String to Parsed Type

- *Haskell has function read*
- *If a type is an instance of the Read type class, read can convert a String which can parse to a value of type to the value*
- *See file wordsRead.hs*

Change type to String

- *Haskell has function show which converts a value of a type which is an instance of Show to a String*
- *For example*

  ```
  putStrLn $ show 5
  ```
- *see file showEx.hs*