Haskell and Functional Programming Basics

2015/2016 1st Semester

CSIS0259 / COMP3259
Principles of Programming Languages

Goal of this Lecture

- To give you a crash course on the basics of Haskell and Functional Programming
- This will be needed for the course, since we will be using Haskell as the implementation Language

Suggested Reading

- Learn You a Haskell for Great Good! (up to Chapter 6)
    - http://learnyouahaskell.com

recommended!

- Haskell tutorial (up to Section 4)
    - http://www.haskell.org/tutorial/

Functional Programming

- What is functional programming?

- PS: I won't be assuming that you are taking the Functional Programming class. So do feel free to leave if you have done functional programming!

Functional Programming

- What is functional programming? Some possible answers:
  - Programming with first-class functions
    - map (\x -> x + 1) [1,2,3] ~> [2,3,4]
  - Programming with mathematical functions
    - No side-effects (no global mutable state, no IO)
    - Calling a function with the same arguments, always returns the same output (not true in most languages!)
  - The main means of computation is function application

Pure Functional Programming

Functional Programming Languages

- Impure Functional Languages
  - Statically Typed: ML,OCaml,Scala …
  - Dynamically Typed: Scheme, Lisp …
- Pure Functional Languages
  - Statically Typed: Haskell

Traditionally focused on Functional Programming

Functional Programming Languages

- Impure Functional Languages
  - Statically Typed: ML,OCaml,Scala, Java 8, C#, C++11, Swift …
  - Dynamically Typed: Scheme, Lisp, Python, Ruby …
- Pure Functional Languages
  - Statically Typed: Haskell, Agda, Idris

Bleeding Edge!

Haskell in the course

- Haskell is going to be used in the course as the implementation language
  - to implement interpreters for programming languages
  - to serve as an example of a (state-of-the-art) functional language

Why Haskell?

- Reason for Haskell in a Programming Languages course:
  - Functional Languages have excellent mechanisms to implement programming languages:
    - Datatypes & Pattern Matching

Installing Haskell

- Haskell Platform (for Windows, mac and Linux)

  http://www.haskell.org/platform/

- IDE/Editor
  - Emacs (recommended)
  - FP Haskell Center (commercial)

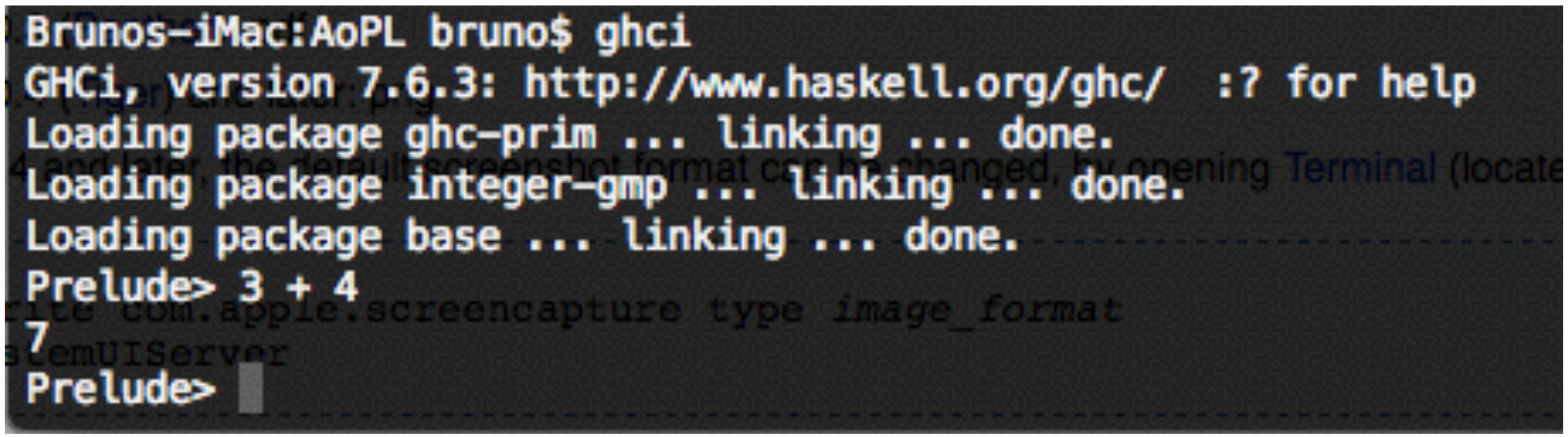
Using Haskell

- The Haskell Platform installs the GHC compiler, which comes with a number of useful tools:
  - ghc is a compiler, similar to gcc
  - ghci is an interactive interpreter
    - great for debugging and testing programs

ghci

- Usage from the command line
  - ghci
  - ghci <filename.hs>
- Usage from emacs
  - Press Control - C - L when in Haskell mode

ghci



Basic commands

- :? (help and available commands)
- :l <file> (load file)
- :r (reload file)
- :set +t (show type information)
- :t e (show the type of an expression)

Basics of Haskell
Demo

Pattern Matching

Pattern Matching

Haskell programs are often defined using pattern matching:

```
hd :: [a] -> a
hd [] = error "cannot take the head of an empty list!"
hd (x:xs) = x
```

To take the head of a list we need to consider two possibilities:

`x :: a`

`xs :: [a]`

1. `[]` – The list is empty
2. `(x:xs)` – The list has one element **`x`** and a tail **`xs`**.

Pattern Matching

Pattern matching can be used on many types:

```
first :: (a,b) -> a
first (x,y) = x

isZero :: Int -> Bool
isZero 0 = True
isZero n = False
```

Here we have used two different types of patterns:

1. **`(x,y)`** – A pattern on a tuple with 2 elements **`x`** and **`y`**.
2. **`0`** – A pattern on numbers.

Pattern Matching Demo

Recursion

Recursion

Haskell programs are often defined using recursion:

```
countElements :: [a] -> Int
countElements []     = 0
countElements (x:xs) = 1 + countElements xs
```

recursive
call

Good recursive definitions normally have:

- Base case(s): Cases where the program terminates.
  - For countElements the base case is `[]`.

- Recursive case(s): Cases where a step is taken towards the base cases.
  - For countElements the recursive case is (`x:xs`).
  - Recursive calls are normally done on **smaller** values.

Recursion Demo

Recursion

There may be programs with bad recursion:

why is this bad?

```
badcountElements1 :: [a] -> Int
badcountElements1 []     = 0
badcountElements1 xs     = 1 + badcountElements1 xs
```

why is this bad?

```
badcountElements2 :: [a] -> Int
badcountElements2 (x:xs) = 1 + badcountElements2 xs
```

Recursion

There may be programs with bad recursion:

```
badcountElements1 :: [a] -> Int
badcountElements1 []      = 0
badcountElements1 xs      = 1 + badcountElements1 xs
```

recursive call is not smaller!

```
badcountElements2 :: [a] -> Int
badcountElements2 (x:xs) = 1 + badcountElements2 xs
```

no base case!

Datatypes

Datatypes

- How are Haskell lists defined? Conceptually they correspond to the following datatype:

```
data [a] = [] | a : [a] — pseudo-code
```

- Note: Haskell lists are actually built-in. The above definition is just for illustration purposes.

User-Defined Datatypes

- Haskell supports user-defined datatypes:

recursive definition

```
data ListInt = Nil | Cons Int ListInt
```

- New datatype definitions support their own pattern matching notation.

```
headListInt :: ListInt -> Int
headListInt Nil = error "Empty list!"
headListInt (Cons x xs) = x
```

Showing and Equality

- Some operations are useful for various datatypes
  - Examples: equality and conversion to a string
- Haskell provides an easy mechanism for supporting these operations:

```
data ListInt = Nil | Cons Int ListInt
                     deriving (Eq,Show)
```

Showing and Equality

- Using equality and show:

```
test :: ListInt -> ListInt -> String
test l1 l2 = if (l1 == l2) then
                  show l1
                else
                  “Not equal!”
```

Demo

Suggested Reading

- Learn You a Haskell for Great Good! (up to Chapter 6)
  - http://learnyouahaskell.com

recommended!

- Haskell tutorial (up to Section 4)
  - http://www.haskell.org/tutorial/

Notes

- Thursday 12:30: tutorial in CPD-3.41
  - May want to bring laptops if you have trouble installing Haskell/emacs
  - Tutorial introducing Haskell and (optional) assignment.