Recollecting Haskell, Part I  
(Based on Chapters 1 and 2 of LYAH*)

CIS 351/Spring 2015  
Programming Languages  
January 13, 2015

*LYAH = Learn You a Haskell for Great Good

---

You can read LYAH  
You will read LYAH.

---

The Blurb from www.haskell.org

Haskell is an advanced purely-functional programming language.  
. . . it allows rapid development of robust, concise, correct software.  

With strong support for  
- integration with other languages,  
- built-in concurrency and parallelism,  
- debuggers,  
- profilers,  
- rich libraries and  
- an active community,

Haskell makes it easier to produce flexible, maintainable, high-quality software.

---

So why do we care about Haskell in this course?

- Haskell is great for prototyping.  
- Forces you to think compositionally.  
- Semi-automated testing: QuickCheck  
- Haskell can give you executable specifications.  
- Good for “model building”  
e.g., direct implementations of operational semantics  

. . . and beyond this course

- Many of the new systems/applications languages  
  (e.g., Swift and Rust) steal lots of ideas from Haskell and ML.  
- These ideas are a lot clearer in Haskell and ML  
  than the munged versions in Swift, Rust, etc.,
Set up

- Go visit http://www.haskell.org.
- Click Download Haskell
- Download the appropriate version of the current Haskell Platform and install it.

!!! Do the above even if you have an old copy of the Haskell Platform from a previous year. You want the latest version of the GHC compiler and libraries.

!!! Use a reasonable editor.

Using Notepad or Word is a waste of your time.


A sample session: ghci as a calculator

```
[Post:] jimroyer% ghci
GHCi, version 7.8.3: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> 2 + 3
5
Prelude> 2 * 3
6
Prelude> 2 - 3
-1
Prelude> 2 / 3
0.6666666666666666
Prelude> :q
Leaving GHCi.
[Post:] jimroyer%
```

Fussy bits

| ✔ | 5 * (-3) |
| ✔ | 5 * 10 - 49 ≡ 5 * (10 - 49) |
| ✗ | 5 * True |

```
Prelude> 5 + True
<interactive>:1:1: (What does all this mean?)
  No instance for (Num Bool)
  arising from the literal ‘5’
  Possible fix: add an instance declaration for (Num Bool)
  In the first argument of ‘(+)’, namely ‘5’
  In the expression: 5 * True
  In an equation for ‘it’: it = 5 * True
```
Defining and using functions, continued

```haskell
baby.hs
doubleMe x = x + x
doubleUs x y = 2*x+2*y
doubleUs' x y = doubleMe x + doubleMe y
doubleSmallNumber x = if x > 100 then x else x * 2
doubleSmallNumber' x = (if x > 100 then x else x * 2)+1
conanO’Brien = "It’s a-me, Conan O’Brien!"
```

Lists

- A list: a sequence of things of the same type.
  - `[2,3,5,7,11,13,17,19]` : [Int]
  - `[True,False,False,True]` : [Bool]
  - `['b','o','b','c','a','t']` \(\equiv\) "bobcat" : [Char]
  - `[]` : [a]
  - `[[1],[2],[3],[4,5,6]]` : [[Int]]
  - `[[1],[2],[3]]` \(\times\)
  - `[[2],[3]]` \(\times\)

- If you want to bundle together things of different types, use tuples (e.g., `(2,True,"foo")` \(\ldots\) explained later).

Lists: Building them up

Write them down: \([item_1, item_2, \ldots, item_n]\)
- `[2,3,5,7,11,13,17,19], [], etc.\]

Concatenation: `list_1++list_2`
- `[1,2,3,4]++[10,20,30] \(\sim\) [1,2,3,4,10,20,30]
- "salt"++"potato" \(\sim\) "saltpotato"
- `[]++[1,2,3] \(\sim\) [1,2,3] [1,2,3]++[] \(\sim\) [1,2,3]
- `1++[2,3] \(\sim\) ERROR [1,2]++3 \(\sim\) ERROR

Cons: `item:list`
- `1:[2,3] \(\sim\) [1,2,3] [1,2]:3 \(\sim\) ERROR`
- `[1,2,3]` is syntactic sugar for `1:2:3:[\ldots] \equiv 1:(2:(3:[\ldots]))`
- You can tell `(:)` is important because they gave it such a short name.

Lists: Tearing them down

- `head [1,2,3] \(\sim\) 1`
- `tail [1,2,3] \(\sim\) [2,3]`
- `init [1,2,3] \(\sim\) [1,2]`
- `head [] \(\sim\) ERROR`
- `tail [] \(\sim\) ERROR`
- `last [1,2,3] \(\sim\) 3`
- `last [] \(\sim\) ERROR`
- `init [] \(\sim\) ERROR`
Lists: More operations

- length :: [a] -> Int
- (!!) :: [a] -> Int -> a
- null :: [a] -> Bool
- reverse :: [a] -> [a]
- drop, take :: Int -> [a] -> [a]
- sum, product :: (Num a) => [a] -> a
- minimum, maximum :: (Ord a) => [a] -> a
- elem, notElem :: (Eq a) => a -> [a] -> Bool

Ranges

- \[m..n\] \sim [m,m+1,m+2,...,n]
- \[1..5\] \sim [1,2,3,4,5]
- \[5..1\] \sim []
- \['a'..'k'\] \sim "abcdefghijk"
- \[1..\] \sim [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17, ...

- \[m,p..n\] \sim [m, m+(p-m), m+2(p-m), ... ,n]
  - \[3,5..10\] \sim [3,5,7,9]
  - \[5,4..1\] \sim [5,4,3,2,1]
  - \[9,7..2\] \sim [9,7,5,3]

*Or the “closest” number before \(n\) that is in the sequence.

List comprehensions

Set comprehensions in math (CIS 275 review)

- \{ x | x \in N, x = x^2 \} = \{ 0,1 \}
- \{ x | x \in N, x > 0 \} = the positive integers
- \{ x^2 | x \in N \} = squares
- \{ (x,y) | x \in N, y \in N, x \leq y \}

Suppose we have \( lst = [5,10,13,4,10]\)

- \[2*n+1 \ | \ n \leftarrow \mathit{lst}\] \sim [11,21,27,9,21]
- \[\mathit{even} \ n \ | \ n \leftarrow \mathit{lst}\] \sim [\mathit{False,True,False,True,True}]
- \[ 2*n+1 \ | \ n \leftarrow \mathit{lst}, \mathit{even}\ n, \ n\geq 5 \] \sim [21,21]

Example: Squaring every element of a list

`squares.hs`

```
squares :: [Integer] -> [Integer]
squares xs = [ x*x | x <- xs ]
```

```
squares [1,2,3] =
  \{ xs = [1,2,3] \}
  [ x*x | x <- [1,2,3] ]

  = \{ x=1 \}, \{ x=2 \}, \{ x=3 \}
  [ 1*1 ] ++ [ 2*2 ] ++ [ 3*3 ]

  =
  [1]++[4]++[9]

  =
  [1,4,9]
```

Example lifted from Phil Wadler.
Example: Odd elements of a list

```haskell
odds :: [Integer] -> [Integer]
odds xs = [ x | x <- xs, odd x ]
```

```haskell
odds [1,2,3]
= [ x | x <- [1,2,3], odd x ]
= [ 1 | odd 1 ] ++ [ 2 | odd 2 ] ++ [ 3 | odd 3 ]
= [ 1 | True ] ++ [ 2 | False ] ++ [ 3 | True ]
= [1,3]
```

*Example lifted from Phil Wadler.*

Example: Sum of the squares of the odd elements, 1

```haskell
sumSqsOdds.hs
squares :: [Integer] -> [Integer]
squares xs = [ x*x | x <- xs ]
```

```haskell
odds :: [Integer] -> [Integer]
odds xs = [ x | x <- xs, odd x ]
of :: [Integer] -> Integer
of xs = sum (squares (odds xs))
f' :: [Integer] -> Integer
f' xs = sum [ x*x | x <- xs, odd x ]
```

Another example lifted from Phil Wadler.

Example: Sum of the squares of the odd elements, 2

```haskell
f xs = sum (squares (odds xs))
```

```haskell
f [1,2,3]
= sum (squares (odds [1,2,3]))
= sum [1,9]
= 10
```

Example: Sum of the squares of the odd elements, 3

```haskell
f' xs = sum [ x*x | x <- xs, odd x ]
```

```haskell
f' [1,2,3]
= sum [ x*x | x <- [1,2,3], odd x ]
= sum [1*1 | odd 1 ] ++ [ 2*2 | odd 2 ] ++ [ 3*3 | odd 3 ]
= sum [1 | True ] ++ [ 4 | False ] ++ [ 9 | True ]
= [1,9]
= 10
```
Example: Sum of the squares of the odd elements, 4

sumSqOdds.hs

```hs
import Test.QuickCheck

squares :: [Integer] -> [Integer]
squares xs = [ x*x | x <- xs ]

odds :: [Integer] -> [Integer]
odds xs = [ x | x <- xs, odd x]

f :: [Integer] -> Integer
f xs = sum (squares (odds xs))

f' :: [Integer] -> Integer
f' xs = sum [ x*x | x <- xs, odd x ]

f_prop :: [Integer] -> Bool
f_prop xs = f xs == f' xs
```

Example: Sum of the squares of the odd elements, 5

```hs
*Main> :load sumSqOdds
[1 of 1] Compiling Main ( sumSqOdds.hs, interpreted )
Ok, modules loaded: Main.

*Main> quickCheck f_prop
+++ OK, passed 100 tests.

*Main>
```

Tuples

Cartesian products in math (More CIS 275 Review)

Suppose \(A, B, C, \ldots\) are sets.

\[
A \times B = \{ (a,b) \mid a \in A, b \in B \}.
\]

\[
A \times B \times C = \{ (a,b,c) \mid a \in A, b \in B, c \in C \}.
\]

etc.

Tuple types are Haskell's version of Cartesian products

- \((1,2)\) :: (Int,Int)
- \((\text{True},'a')\) :: (Bool,Char)
- \((3,'q','foo')\) :: (Int,Char,[Char])
- \([1,2],(3,4,5),(6,7)\) (Why?)
- \([1,2],('d',False)\) (Why?)

Pairs

- \(\text{fst} : (a,b) \rightarrow a\)
  \(\text{fst} ("muffin",99) \Rightarrow "muffin"
- \(\text{snd} : (a,b) \rightarrow b\)
  \(\text{snd} ("muffin",99) \Rightarrow 99\)

- \(\times\)
  \(\text{fst (4.1,True,'a')} \Rightarrow \text{ERROR}\)
  \(\text{snd (4.1,True,'a')} \Rightarrow \text{ERROR}\)

- \(\text{zip} : [a] \rightarrow [b] \rightarrow [(a,b)]\)
  \(\text{zip [1..5]} [\text{'a'},\text{'b'},\text{'c'},\text{'d'},\text{'e'}]\)
  \(\Rightarrow [(1,\text{'a'}),(2,\text{'b'}),(3,\text{'c'}),(4,\text{'d'}),(5,\text{'e'})]\)
  \(\text{zip [1..]} \) "abcde"
  \(\Rightarrow [(1,\text{'a'}),(2,\text{'b'}),(3,\text{'c'}),(4,\text{'d'}),(5,\text{'e'})]\)
Finding Pythagorean Triples

Problem
Find all possible values of \((a, b, c)\) such that \(a\), \(b\), and \(c\) are integers \(\leq 10\) that are the edge-lengths of a right triangle with perimeter 24.

- \(\text{triples1} = [(a, b, c) | a \leftarrow [1..10], b \leftarrow [1..10], c \leftarrow [1..10]]\)
- \(\text{triples2} = [(a, b, c) | a \leftarrow [1..10], b \leftarrow [a..10], c \leftarrow [b..10]]\)
- \(\text{triples3} = [(a, b, c) | a \leftarrow [1..10], b \leftarrow [a..10], c \leftarrow [b..10], a^2 + b^2 = c^2]\)
- \(\text{triples4} = [(a, b, c) | a \leftarrow [1..10], b \leftarrow [a..10], c \leftarrow [b..10], a^2 + b^2 = c^2, a+b+c = 24]\)