

HASKELL BASICS AND TYPES

Curt Clifton

Rose-Hulman Institute of Technology

Check out *Haske11Basics* from SVN



MORE BASICS

HASKELL IS: LAZY

- No computation takes place unless it is forced to when the result is used
- Let's us make infinite lists!
 - Example: *makeList = 1 : makeList*
- Useful function from the Prelude: *take n xs*
- Try writing: *upFrom n*
 - Example: *upFrom 5* yields *[5,6,7,8,...]*

HASKELL IS: CASE SENSITIVE

- Functions **must** start with lower case
- Types **must** start with upper case
- More info. on types coming...

Mostly

HASKELL IS: PURELY FUNCTIONAL

- Given the same arguments, a function in Haskell always produces the same results
- Sometimes referred to as *referential transparency*
- This allows automatic *memoization*
 - Storing the results of previously evaluated functions
- Mostly? Impurity needed for I/O and persistence

HASKELL IS: STRONGLY, STATICALLY TYPED

- All types must be given or inferable (guess-able) at “compile” time
- *Type inference*: known (or inferred) types of functions and arguments are used to infer types of other arguments and functions
- Try this:
 :t | | + 3 |
 :t | | + 3 | :: Int
 :t (+)

'IF' IS AN EXPRESSION

- $myDrop\ n\ xs = if\ n\ \leq\ 0\ ||\ null\ xs$
 $then\ xs$
 $else\ myDrop\ (n - 1)\ (tail\ xs)$
- Can't have a one-legged *if* in Haskell. Why not?

FUN WITH LISTS

- What is the type of *map*, *filter*, *foldr*, *foldl*, *zip*, *zipWith*?
- Try:
 - Add *import List* to top of your *basics.hs* file
 - Reload, then enter
 - *:browse List*
 - *:info filter*
- Recall: *[1..10]* yields *[1,2,3,4,5,6,7,8,9,10]*

Also see <http://www.haskell.org/ghc/docs/latest/html/libraries/>

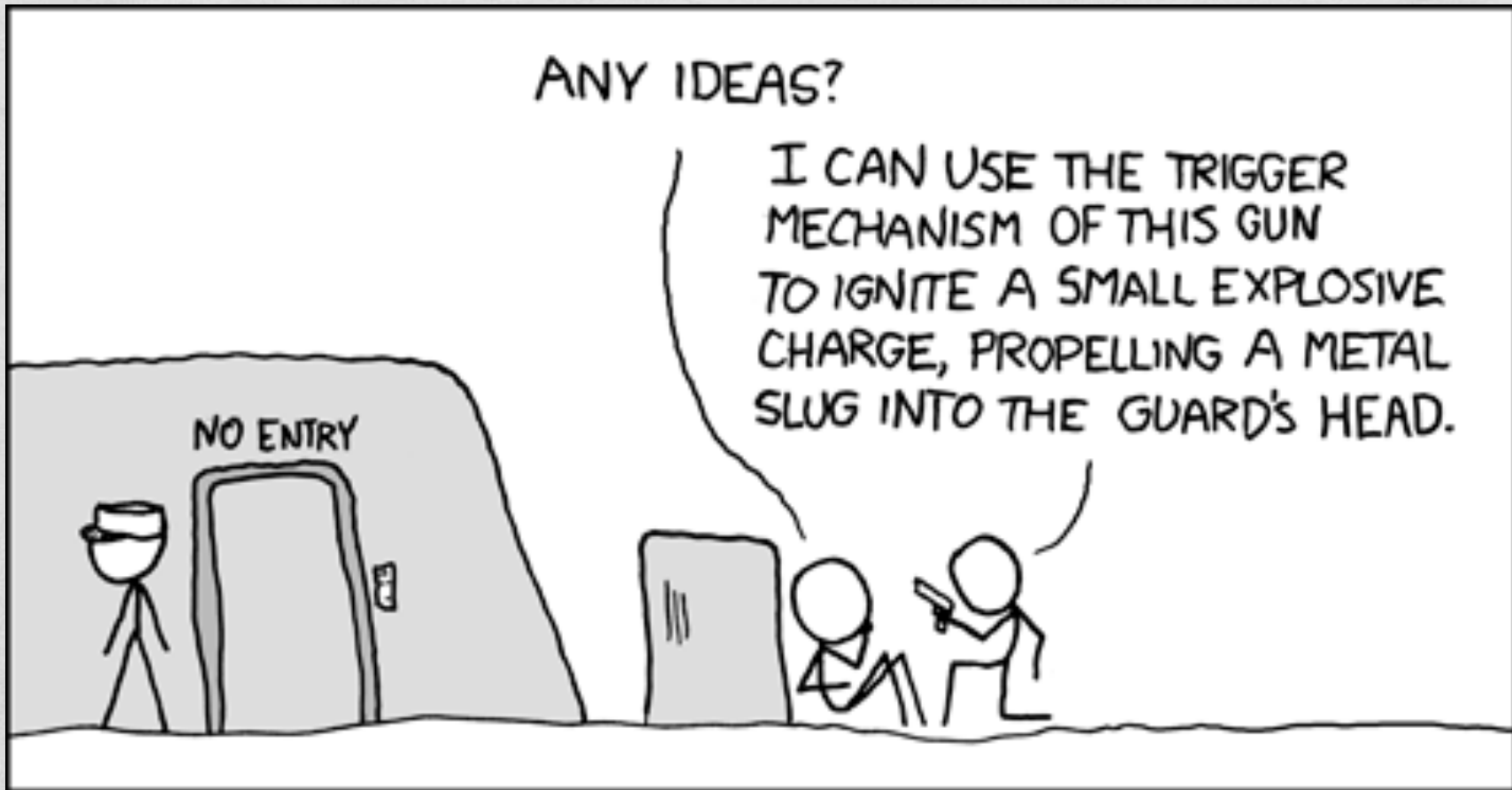
LAZY FIB

Gives the n^{th}
element of *fibList*

- *fastFib* $n = \text{fibList} !! n$
where *fibList* = $0 : 1 : \text{zipWith } (+) \text{ fibList } (\text{tail fibList})$

Parentheses turn
infix operator into
a function

MACGYVER GETS LAZY



<http://xkcd.com/444/>

At the time of this writing, Wikipedia has a wonderful article titled 'List of problems solved by MacGyver'.



TYPES

READING HASKELL TYPES

- *:type logBase → logBase :: (Floating a) => a -> a -> a*
- Read: “given that *a* is a Floating point type, then *logBase* is a function that takes two arguments of type *a* and returns a value of type *a*”

DECLARING TYPES OF FUNCTIONS

- We can declare specific types for functions:
 - $upFrom :: (Num a) \Rightarrow a \rightarrow [a]$
- Why useful?
- Helpful hint for learning types:
 - Make ghci display the type of each result by entering: `:set +t`
 - Add it to `ghci.conf` if you want

TYPE SYNONYMS

- Type synonyms let us give additional names to existing types → improves readability
- *type BookID = Int*
type Title = String
type Author = String

DECLARING CUSTOM DATA TYPES

keyword

```
data BookInfo = Book BookID Title [Author]
```

custom
type name

constructor
definition

types of constructor
parameters

Try: `:t Book`

`:t Book 1 23 "Little Schemer" ["Friedman", "Felleisen"]`

CUSTOM DATA TYPES

- Use constructors to make values with the custom type

```
>>> Book 123 "Is" ["f","f"]
```

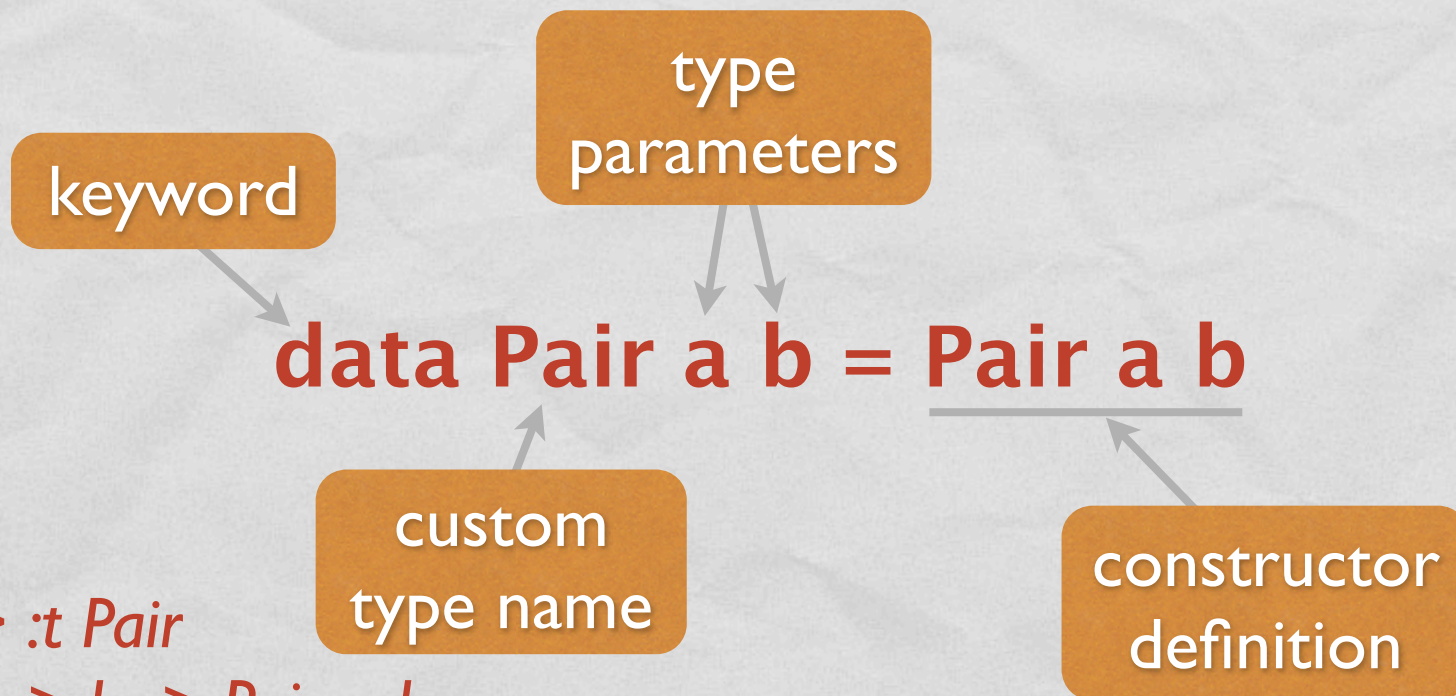
- Can make custom types *instances of type classes*

```
data BookInfo = Book ...  
  deriving (Show)
```

- Can pattern match against the types

```
title (Book _ t _) = t  
firstAuth (Book _ _ (x:_)) = x
```


POLYMORPHIC CUSTOM DATA TYPES



```
*Main> :t Pair
```

```
Pair :: a -> b -> Pair a b
```

```
*Main> :t Pair 'c' "Saw"
```

```
Pair 'c' "Saw" :: Pair Char [Char]
```

```
*Main> :t Pair 1 'c'
```

```
Pair 2 'c' :: (Num t) => Pair t Char
```

type name and constructor name can be the same

CONSIDER...

- Consider:

findElement :: (a -> Bool) -> [a] -> a

findElement p (x:xs) =

if p x

then x

else findElement p xs

- What should we do if we don't find a match?

MULTIPLE CONSTRUCTORS AND THE *MAYBE* TYPE

- The Haskell Prelude defines a custom type:
 - *data Maybe a = Nothing
| Just a*
- Example:
 - *findElement2 :: (a -> Bool) -> [a] -> Maybe a
findElement2 _ [] = Nothing
findElement2 p (x:xs) =
if p x
then Just x
else findElement2 p xs*