“There are two ways of constructing software. One way is to make it so simple, that there are obviously no deficiencies, The other way is to make it so complicated that there are no obvious deficiencies.”

Goal: Obviously No Deficiencies

Readable

Reusable
Goal: Obviously No Deficiencies

Modifiable

Predictable

Goal: Obviously No Deficiencies

Checkable

Yes, but how?
Goal: Obviously No Deficiencies

Functional Programming(?)

No Assignment.
No Mutation.
No Loops.

Functional Programming?
Readable
Reusable
Modifiable
Predictable
Checkable

“You’ve got to be kidding me!”

John McEnroe
Wimbledon, 1980
So, Who Uses FP?

PL Researchers.

Functional Programming?
- Readable
- Reusable
- Modifiable
- Predictable
- Checkable
- Parallelizable

Google
MapReduce

Microsoft
F#
So, Who Uses FP?

- Erlang
- Scala
- Wall Street
- CSE 230
CSE 230: Medium of Instruction

Haskell

Why Haskell?

Bleeding edge PL.

Why Haskell?

Blows Your Mind.

Why Haskell?

Beautiful.
Why Haskell?

Alan Perlis
Epigrams In Programming

“A language that doesn't affect how you think about programming, isn’t worth knowing”

Why Haskell?

Fun.

I wanted to learn it.

CSE 230: Outline

1. FP & Abstraction
   Readable
   Reusable

2. Types & Analysis
   Modifiable
   Predictable
   Checkable
CSE 230 : Personnel

Instructor
Ranjit Jhala (jhala@cs)

TA
Pat Rondon (prondon@cs)

CSE 230 : Materials

Web
http://cseweb.ucsd.edu/classes/wi11/cse230

Board
http://webct.ucsd.edu

Book
Haskell School of Expression (SOE)

CSE 230 : Grading

[10%] Class Participation

[60%] Pair Assignments

[30%] Take-home Final
What is Haskell?

Programming in Haskell

“Computation by Calculation”

What is Haskell?

Programming in Haskell

“Substitute Equals by Equals”
Substituting Equals

\[
3 \times (4 + 5) \downarrow \\
3 \times 9 \downarrow \\
27
\]

That’s it!

What is Abstraction?
Pattern Recognition

Pattern Recognition

\[
\text{pat } x \ y \ z = x \times (y + z)
\]

\[
\begin{align*}
\text{pat } 31 \ 42 \ 56 &= 31 \times (42 + 56) \\
\text{pat } 70 \ 12 \ 95 &= 70 \times (12 + 95) \\
\text{pat } 90 \ 68 \ 12 &= 90 \times (68 + 12)
\end{align*}
\]

Pattern Application: “Fun Call”

\[
\text{pat } x \ y \ z = x \times (y + z)
\]

\[
\begin{align*}
\text{pat } 31 \ 42 \ 56 \downarrow \\
31 \times (42 + 56) \downarrow \\
31 \times 98 \downarrow \\
3038
\end{align*}
\]
Programming in Haskell
“Substitute Equals by Equals”
Really, that’s it!

Elements of Haskell
Expressions, Values, Types

Expressions

Values
Types

expression :: Type

value :: Type

---

The GHC System

Batch Compiler “ghc”
Compile & Run Large Programs

Interactive Shell “ghci”
Tinker with Small Programs

---

Interactive Shell: ghci

:load foo.hs
:type expression
:info variable
### Basic Types

- $31 \times (42 + 56) :: \text{Integer}$
- $3 \times (4.2 + 5.6) :: \text{Double}$
- ‘a’ :: Char
- True :: Bool

**Note:** + and * overloaded ...

### Function Types

- $A \rightarrow B$

**Function** taking input of $A$, yielding output of $B$

- pos :: Integer -> Bool
  
  pos $x = (x > 0)$

### "Multi-Argument" Function Types

- $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow B$

**Function** taking args of $A_1$, $A_2$, $A_3$, giving out $B$

- pat :: Int -> Int -> Int -> Bool
  
  pat $x$ $y$ $z = x \times (y + z)$

### Tuples

- $(A_1, \ldots, A_n)$

**Bounded Sequence** of values of type $A_1, \ldots, A_n$

- (‘a’, 5) :: (Char, Int)
- (‘a’, 5.2, 7) :: (Char, Double, Int)
- ((7, 5.2), True) ::
Extracting Values From Tuples

Pattern Matching extracts values from tuple

\[(A_1, A_2, \ldots, A_n)\]

\[
\text{pat} :: \text{Int} \to \text{Int} \to \text{Int} \to \text{Bool}
\]

\[
\text{pat} \ x \ y \ z = x \times (y + z)
\]

\[
\text{pat}' :: (\text{Int}, \text{Int}, \text{Int}) \to \text{Int}
\]

\[
\text{pat}' \ (x, y, z) = x \times (y + z)
\]

Lists

Unbounded Sequence of values of types \(A\)

\[
[\text{a',b',c'}] ::
\]

\[
[1,3,5,7] ::
\]

\[
[(1,\text{True}),(2,\text{False})] ::
\]

\[
[[1],[2,3],[4,5,6]] ::
\]

List’s Values Must Have Same Type

Unbounded Sequence of values of types \(A\)

\[
[1, 2, ‘c’] :: [\text{A}]
\]

What is \(A\) ?

(Mysterious) Type Error!
“Cons”tructing Lists

\((::) : : \text{a} \rightarrow [\text{a}] \rightarrow [\text{a}]\)

Input: element ("head") and list ("tail")

Output: new list with head followed by tail

\('a' : ['b', 'c'] \Rightarrow ['a', 'b', 'c']\)

\(1 : [] \Rightarrow [1]\)

\([] : [] \Rightarrow\)

Syntactic Sugar

\([x_1, x_2, \ldots, x_n]\)

Is actually a pretty way of writing

\(x_1 : x_2 : \ldots : x_n : []\)

“Cons”tructing Lists

\(\text{cons2} ::\)

\(\text{cons2} \ x \ y \ zs = x : y : zs\)

\(\text{cons2} \ 'a' \ 'b' \ [c'] \Rightarrow ['a', 'b', 'c']\)

\(\text{cons2} \ 1 \ 2 \ [3,4,5,6] \Rightarrow [1,2,3,4,5,6]\)

Function Practice : List Generation

\(\text{clone} :: \text{a} \rightarrow \text{Int} \rightarrow [\text{a}]\)

\(\text{clone} \ x \ n = \text{if} \ n==0\)

\(\quad \text{then} \ []\)

\(\text{else} \ x : (\text{clone} x (n-1))\)

\(\text{clone} \ 'a' \ 4 \Rightarrow ['a', 'a', 'a', 'a']\)

\(\text{clone} \ 1.1 \ 3 \Rightarrow [1.1, 1.1, 1.1]\)
clone :: a -> Int -> [a]
clone x 0 = []
clone x n = x: (clone x (n-1))

Define with multiple equations
More Readable

clone 'a' 3
⇒ 'a': (clone 'a' 2)
⇒ 'a': ('a': (clone 'a' 1))
⇒ 'a': ('a': ('a': (clone 'a' 0)))
⇒ ['a': ('a': ('a': (clone 'a' 0))))

Ugly, Complex Expression

clone :: a -> Int -> [a]
clone x 0 = []
clone x n = let tl = clone x (n-1) in x:tl

Define with local variables
More Readable
**Function Practice : List Generation**

clone :: a -> Int -> [a]
clone x 0 = []
clone x n = x:tl
  where tl = clone x (n-1)

Define with local variables
More Readable

range :: Int -> Int -> [Int]
range i j = if i<=j
  then []
  else i:(range (i+1) j)

range 2 8  ⇒  [2,3,4,5,6,7,8]

Define with multiple guards
More Readable

listAdd :: [Integer] -> Integer
listAdd []     = 0
listAdd (x:xs) = x + listAdd xs

Access elements By Pattern Matching
listAdd []     = 0
listAdd (x:xs) = x + listAdd xs
Recap

Execution = Substitute Equals

Expressions, Values, Types

Base Vals, Tuples, Lists, Functions

Next: Creating Types

Type Synonyms

Names for Compound Types

type XY = (Double, Double)

Not a new type, just shorthand

Write types to represent:

Circle: x-coord, y-coord, radius

type Circle = (Double, Double, Double)

Square: x-coord, y-coord, side

type Square = (Double, Double, Double)
Type Synonyms

Bug Alarm!
Call areaSquare on circle, get back junk

type Circle = (Double, Double, Double)
    areaCircle (_,_,r) = pi * r * r

type Square = (Double, Double, Double)
    areaSquare (_,_,d) = d * d

Solution: New Data Type

data CircleT = Circle (Double, Double, Double)
data SquareT = Square (Double, Double, Double)

Creates New Types
CircleT
SquareT

Creates New Constructors
Circle :: (Double, Double, Double) -> CircleT
Square :: (Double, Double, Double) -> SquareT

Only way to create values of new type

Solution: New Data Type

data CircleT = Circle (Double, Double, Double)
data SquareT = Square (Double, Double, Double)

Creates New Constructors
Circle :: (Double, Double, Double) -> CircleT
Square :: (Double, Double, Double) -> SquareT

How to access/deconstruct values?
Deconstructing Data

areaSquare :: CircleT -> Double
areaCircle (Circle(_,_,r)) = \( \pi \times r \times r \)

areaSquare :: SquareT -> Double
areaSquare (Square(_,_,d)) = \( d \times d \)

How to access/deconstruct values?
Pattern Match…!

Call areaSquare on CircleT?
Different Types: GHC catches bug!

How to build a list with squares & circles?
Restriction: List elements have same type!

Solution: Create a type to represent both!
Variant (aka Union) Types

Create a type to represent both!

data CorS =
  | Circle (Double,Double,Double)
  | Square (Double,Double,Double)

  Circle(1,1,1) :: CorS
  Square(2,3,4) :: CorS

  [Circle(1,1,1), Square(2,3,4)] :: [CorS]

Variant (aka Union) Types

Access/Deconstruct by Pattern Match

data CorS =
  | Circle (Double,Double,Double)
  | Square (Double,Double,Double)

  area :: CorS -> Double
  area (Circle(_,_,r)) = pi*r*r
  area (Square(_,_,d)) = d*d

A Richer Shape

Lets drop the parens...

data Shape =
  | Rectangle (Double, Double)
  | Ellipse   (Double, Double)
  | RtTriangle(Double, Double)
  | Polygon   [(Double, Double)]

A Richer Shape

Lets drop the parens...

data Shape =
  | Rectangle  Double Double
  | Ellipse    Double Double
  | RtTriangle Double Double
  | Polygon    [(Double, Double)]
A Richer Shape

\[
data \text{ Shape } = \\
| \text{Rectangle} \quad \text{Double} \quad \text{Double} \\
| \text{Ellipse} \quad \text{Double} \quad \text{Double} \\
| \text{RtTriangle} \quad \text{Double} \quad \text{Double} \\
| \text{Polygon} \quad [(\text{Double}, \text{Double})] \\
\]

Why can’t we drop last case’s parens?

Making Shape Readable

\[
data \text{ Shape } = \\
| \text{Rectangle} \quad \text{Side} \quad \text{Side} \\
| \text{Ellipse} \quad \text{Radius} \quad \text{Radius} \\
| \text{RtTriangle} \quad \text{Side} \quad \text{Side} \\
| \text{Polygon} \quad [\text{Vertex}] \\
\]

type \text{ Side } = \text{Double} \\
type \text{ Radius } = \text{Double} \\
type \text{ Vertex } = (\text{Double}, \text{Double})

Calculating The Area

\[
\text{area} :: \text{ Shape } \rightarrow \text{ Double} \\
\text{area} (\text{Rectangle} \ l \ b) = l \times b \\
\text{area} (\text{RtTriangle} \ b \ h) = b \times h / 2 \\
\text{area} (\text{Ellipse} \ r1 \ r2) = \pi \times r1 \times r2
\]

GHC warns about missing case!

Calculating Area of Polygon

\[
\text{area} (\text{Polygon} (\text{v1}:\text{v2}:\text{v3}:\text{vs})) = \text{triArea} \text{v1} \text{v2} \text{v3} + \text{area} (\text{Polygon} (\text{v1}:\text{v3}:\text{vs})) \\
\text{area} (\text{Polygon} _) = 0
\]
“Hello World”
Input/Output in Haskell

Programs Interact With The World
(Don’t just compute values!)

Programs Interact With The World
Read files,
Display graphics,
Broadcast packets, ...

Programs Interact With The World
How to fit w/ values & calculation ?
I/O via an “Action” Value

Action
Value describing an effect on world

IO a
Type of an action that returns an a

Example: Output Action

Just do something, return nothing

putStr :: String -> IO ()
takes input string, returns action that writes string to stdout

Example: Output Action

Only one way to “execute” action
make it the value of name main

main :: IO ()
main = putStr “Hello World! \n”

Example: Output Action

Compile and Run

ghc -o hello helloworld.hs

main :: IO ()
main = putStr “Hello World! \n”
"Execute" in ghci
:load helloworld.hs

main :: IO ()
main = putStrLn "Hello World! \n"

Writing does not trigger Execution

act2 :: (IO (), IO ())
act2 = (putStrLn "Hello", putStrLn "World")

Just creates a pair of actions...

How to do many actions?
By composing small actions

main :: IO ()
Just “do” it

\begin{verbatim}
do putStrLn "Hello"
putStrLn "World"
putStrLn "\n"
\end{verbatim}

Single Action
“Sequence” of sub-actions

Just “do” it

\begin{verbatim}
do act1
act2
...
actn
\end{verbatim}

Block Begin/End via Indentation
“Offside Rule” (Ch3. RWH)

Example: Input Action

Action that returns a value

\texttt{getLine :: IO String}

Read and Return Line from StdIn
Example: Input Action

Name result via “assignment”

\[
\begin{align*}
x & \leftarrow \text{act} \\
x \text{ refers to result in later code}
\end{align*}
\]

Example: Input Action

Name result via “assignment”

\[
\begin{align*}
\text{main} & \colon \text{IO ()} \\
\text{main} & = \text{do} \ \text{putStr} \ “\text{What is your name?}” \\
& \quad \ n \leftarrow \text{getLine} \\
& \quad \text{putStrLn} \ (“\text{Happy New Year }” ++ n)
\end{align*}
\]