CSEN403: Concepts of Programming Languages
Functional Programming V: A Wrap

Prof. Dr. Slim Abdennadher
Dr. Nada Sharaf

Spring Semester 2021

Data Types

```
type Number = Int
type Point = (Number,Number)
type Length = Number
data Shape = Pt Point
       | Circle Point Length
       | Rect Point Length Length
      deriving (Eq,Show)
type Figure = [Shape]
type BBox = (Point,Point)
```

- A circle is defined by the center point and the raduis
- A point is defined by its x and y coordinates
- A rectangle is defined by the lower left corner, its width and length

Data Types

```
type Number = Int
type Point = (Number,Number)
type Length = Number
data Shape = Pt Point
       | Circle Point Length
       | Rect Point Length Length
      deriving (Eq,Show)
type Figure = [Shape]
type BBox = (Point,Point)
```

- A circle is defined by the center point and the raduis
- A point is defined by its x and y coordinates
- A rectangle is defined by the lower left corner, its width and length

- Define a function minX that computes the minimum x-coordinate of a shape
- Define a function move that moves the position of a shape by a vector given by a point.

Solution

- ```
  minX (Pt (x,y)) = x
  minX (Circle (x,y) r) = x-r
  minX (Rect (x,y) w h) = x
  ```
  ```

Solution

- ```
  minX (Pt (x,y)) = x
  minX (Circle (x,y) r) = x-r
  minX (Rect (x,y) w h) = x
  ```

- ```
  move (Pt(x,y)) (px,py)= Pt(x+px,y+py)
  move (Circle (x,y) r) (px,py) = (Circle (x+px,y+py) r)
  move (Rect (x,y) w h) (px,py)= (Rect (x+px,y+py) w h)
  ```

Types

```
mystery i p f =
        if p i then i: mystery (f i) p f
        else []
```

Types

```
mystery i p f =
        if p i then i: mystery (f i) p f
        else []
```

- What is the type of `mystery`

Types

```
mystery i p f =
        if p i then i: mystery (f i) p f
        else []
```

- What is the type of `mystery`

```
mystery :: a -> (a -> Bool) -> (a -> a) -> [a]
```

Types

```
mystery i p f =
        if p i then i: mystery (f i) p f
        else []
```

- What is the type of `mystery`

  ```
  mystery :: a -> (a -> Bool) -> (a -> a) -> [a]
  ```

- What is the output of `mystery 0 (< 15) (+4)`
  use V177

More Examples

Provide a function to split a list of integers into two lists, such that the first list contains all non-negative integers and the second list contains all negative integers. For instance, the list [1,-2,0,3,-4] should be split into lists [1,0,3] and [-2,-4].

```
>split [1,-2,3,5,-6]
([1,3,5],[-2,-6])
> split [1,3,5]
([1,3,5],[])
```

- Using higher-order functions:

More Examples

Provide a function to split a list of integers into two lists, such that the first list contains all non-negative integers and the second list contains all negative integers. For instance, the list [1,-2,0,3,-4] should be split into lists [1,0,3] and [-2,-4].

```
>split [1,-2,3,5,-6]
([1,3,5],[-2,-6])
> split [1,3,5]
([1,3,5],[])
```

- Using higher-order functions:
  `split l = (filter (>=0) l,filter (<0) l)`

More Examples

Provide a function to split a list of integers into two lists, such that the first list contains all non-negative integers and the second list contains all negative integers. For instance, the list [1,-2,0,3,-4] should be split into lists [1,0,3] and [-2,-4].

```
>split [1,-2,3,5,-6]
([1,3,5],[-2,-6])
> split [1,3,5]
([1,3,5],[])
```

- Using higher-order functions:
  ```
  split l = (filter (>=0) l,filter (<0) l)
  ```
- Recursively

More Examples

Provide a function to split a list of integers into two lists, such that the first list contains all non-negative integers and the second list contains all negative integers. For instance, the list [1,-2,0,3,-4] should be split into lists [1,0,3] and [-2,-4].

```
>split [1,-2,3,5,-6]
([1,3,5],[-2,-6])
> split [1,3,5]
([1,3,5],[])
```

- Using higher-order functions:

```
split l = (filter (>=0) l,filter (<0) l)
```

- Recursively

```
split [] = ([],[])
split (x:xs) = if x>=0 then (x:l1,l2)
               else (l1,x:l2)
               where (l1,l2) = split xs
```

Frame Title

```
map :: (a -> b) -> [a] -> [b]
foldr :: (a -> b -> b) -> b -> [a] -> b
filter :: (a -> Bool) -> [a] -> [a]
```

Haskell in Industry

Haskell in Industry

- *Haxl*: a Haskell library used by Facebook
  - Used for fetching and dealing with remote data
  - is able to cache previous requests
- *Sigma*:
  - Spam Detection

Other Examples

- Prezi
  - Elm language designed for front-end development
- Fynder
  - Uses haskell for both front and back ends

Haskell in Research

- Artificial Intelligence
- Security

Thank you