Records

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Objectives

- Understand the motivation for record types.
- Understand the syntax for declaration and use.

Records

- Complex numbers have the form $a + bi$, where $i \equiv \sqrt{-1}$
- Addition: $(a + bi) + (c + di) = (a + c) + (b + d)i$
- Multiplication: $(a + bi) \times (c + di) = ac - bd + (ad + bc)i$

```
1 cadd (a,b) (c,d) = (a + c, b + d)
2 cmul (a,b) (c,d) = (a * c - b * d,
3                     a * d + b * c)
```

We could use tuples to represent complex numbers, like this. (What are the types of these functions?) Why might this be a bad idea?

```
1 Prelude> :t cadd
2 cadd :: (Num t, Num t1) => (t, t1) -> (t, t1) -> (t, t1)
```

Record Type Definitions

Record Syntax

`data` *Name* `= Name { field :: type [, field :: type ...] }`

```haskell
data Complex = Complex { re :: Float, im :: Float }

cadd x y = Complex { re = re x + re y
                   , im = im x + im y }
cmul x y = Complex { re = re x * re y - im x * im y
                   , im = re x * im y + re y * im x }
```

Each of the field names becomes a function in Haskell. Therefore, the *field names must be unique.*

- To create an element of type complex, you have two choices.
  1. Treat the constructor as a function:

```
c = Complex 10.54 34.2
```

  2. Specify the field names:

```
c = Complex { re = 10.54, im = 34.2 }
```

Haskell creates the field selector functions automatically.

```
Main> re c
10.54
Main> im c
34.2
```

Example: Database Records

- Records are often used to model database-like data.
- Example: we want to store first name, last name, and age.

```
1 data Person = Person { fname :: String
2                      , lname :: String
3                      , age :: Int }
4 people = [ Person "Nathan" "Park" 7,
5            Person "Naomi" "DeMonia" 93 ]
```

- You may want to derive `Show` and `Eq` to be able to print and test for equality.