PROGRAMMING IN HASKELL

Chapter 10.2 - Syntax and Use of Records

First try

We wish to construct a data type that describes a Student.
We want to record the following

- First Name
- Second Name
- Age
- Course
- email address

We could try :

```
data Student = Student String String  Int  String String
               deriving (Show)
```

First try.. Contd.


```
data Student = Student String String  Int  String String
               deriving (Show)
```

Then write functions to return back the various elements, e.g.

```
firstName :: Student -> String
firstName (Student firstname _ _ _ _ ) = firstname

age :: Student -> Int
age (Student _ _ age _ _ )  = age
etc..
```

Then use..


```
ghci> let student1 =
        Student ”Adam" ”Ant" 21 ”Computer Science”
                ”stud1@wit.ie"
ghci> firstName student1
ghci> ” Adam”
```

There is a better way


```
data Student = Student { firstName :: String
                       , lastName :: String
                       , age :: Int
                       , course :: String
                       , email :: String
                       } deriving (Show)
```

There is a better way

Main benefit – we now have created lookup functions so

```
ghci> :t firstName
ghci> Student -> String
```

And using the student1 from before

```
ghci> firstName student1
ghci> ”Adam”
```

To construct a student:


```
student1 = Student {     firstName = "Adam",
                    lastName="Ant",
                    age=21,
                    course="Applied",
                    email="mm"}
```

lookup functions

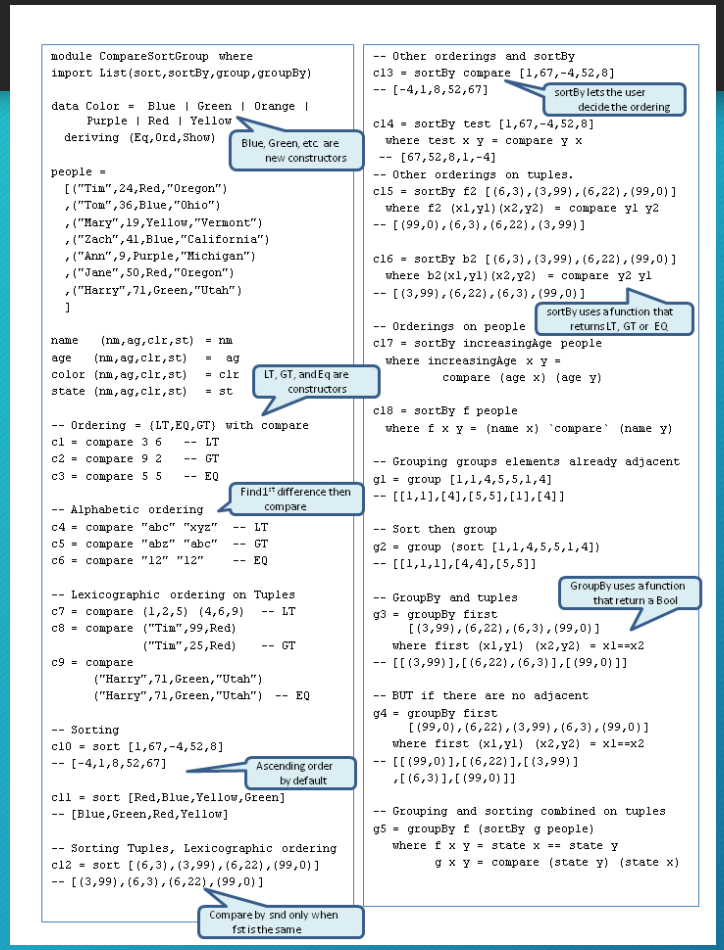
So, given:

```
student1 = Student {    firstName = "Adam",
                        lastName="Ant",
                        age=21,
                        course="Applied",
                        email="mm”
                                }
```

We can access the elements:

```
ghci> lastName student1
ghci> ”Ant"
```

Sorting and grouping on lists of tuples



```
module CompareSortGroup where
import List(sort,sortBy,group,groupBy)

data Color =  Blue | Green | Orange |
     Purple | Red | Yellow
  deriving (Eq,Ord,Show)

people =
  [("Tim",24,Red,"Oregon")
  ,("Tom",36,Blue,"Ohio")
  ,("Mary",19,Yellow,"Vermont")
  ,("Zach",41,Blue,"California")
  ,("Ann",9,Purple,"Michigan")
  ,("Jane",50,Red,"Oregon")
  ,("Harry",71,Green,"Utah")
  ]

name  (nm,ag,clr,st) = nm
age   (nm,ag,clr,st) = ag
color (nm,ag,clr,st) = clr
state (nm,ag,clr,st) = st

-- Ordering = {LT,EQ,GT} with compare
c1 = compare 3 6   -- LT
c2 = compare 9 2   -- GT
c3 = compare 5 5   -- EQ

-- Alphabetic ordering
c4 = compare "abc" "xyz"  -- LT
c5 = compare "abz" "abc"  -- GT
c6 = compare "12" "12"    -- EQ

-- Lexicographic ordering on Tuples
c7 = compare (1,2,5) (4,6,9)  -- LT
c8 = compare ("Tim",99,Red)
             ("Tim",25,Red)   -- GT
c9 = compare
       ("Harry",71,Green,"Utah")
       ("Harry",71,Green,"Utah")  -- EQ

-- Sorting
c10 = sort [1,67,-4,52,8]
-- [-4,1,8,52,67]

c11 = sort [Red,Blue,Yellow,Green]
-- [Blue,Green,Red,Yellow]

-- Sorting Tuples, Lexicographic ordering
c12 = sort [(6,3),(3,99),(6,22),(99,0)]
-- [(3,99),(6,3),(6,22),(99,0)]

-- Other orderings and sortBy
c13 = sortBy compare [1,67,-4,52,8]
-- [-4,1,8,52,67]

c14 = sortBy test [1,67,-4,52,8]
  where test x y = compare y x
 -- [67,52,8,1,-4]
-- Other orderings on tuples.
c15 = sortBy f2 [(6,3),(3,99),(6,22),(99,0)]
  where f2 (x1,y1)(x2,y2) = compare y1 y2
-- [(99,0),(6,3),(6,22),(3,99)]

c16 = sortBy b2 [(6,3),(3,99),(6,22),(99,0)]
  where b2(x1,y1)(x2,y2) = compare y2 y1
-- [(3,99),(6,22),(6,3),(99,0)]

-- Orderings on people
c17 = sortBy increasingAge people
  where increasingAge x y =
          compare (age x) (age y)

c18 = sortBy f people
  where f x y = (name x) `compare` (name y)

-- Grouping groups elements already adjacent
g1 = group [1,1,4,5,5,1,4]
-- [[1,1],[4],[5,5],[1],[4]]

-- Sort then group
g2 = group (sort [1,1,4,5,5,1,4])
-- [[1,1,1],[4,4],[5,5]]

-- GroupBy and tuples
g3 = groupBy first
     [(3,99),(6,22),(6,3),(99,0)]
   where first (x1,y1) (x2,y2) = x1==x2
-- [[(3,99)],[(6,22),(6,3)],[(99,0)]]

-- BUT if there are no adjacent
g4 = groupBy first
     [(99,0),(6,22),(3,99),(6,3),(99,0)]
   where first (x1,y1) (x2,y2) = x1==x2
-- [[(99,0)],[(6,22)],[(3,99)]
   ,[(6,3)],[(99,0)]]

-- Grouping and sorting combined on tuples
g5 = groupBy f (sortBy g people)
   where f x y = state x == state y
         g x y = compare (state y) (state x)
```


http://web.cecs.pdx.edu/~sheard/course/Cs163/Haskell/Crib07.hs


ANY
QUESTIONS?