

Programming assignment 1

Hardy Jones

999397426

Professor Köeppe

This file is a literate haskell file. So it could be run directly if that were required.

We first require some pragmas to let us know if something slipped by.

```
> {-# OPTIONS_GHC -Wall #-}  
> {-# OPTIONS_GHC -Werror #-}  
> {-# OPTIONS_GHC -fno-warn-unused-do-bind #-}
```

And also for lens generation:

```
> {-# LANGUAGE TemplateHaskell #-}
```

Now on to our actual program:

```
> module P1 where
```

We import a whole mess of helper things

```
> import Control.Lens ((^.), (#), makeLenses, to)  
  
> import Data.Geo.Geodetic (Sphere, _Sphere)  
> import Data.Tuple (swap)  
> import Data.Typeable (Typeable)  
  
> import Text.Parsec (anyChar, endOfLine, manyTill, spaces, string, try)  
> import Text.Parsec.String (Parser, parseFromFile)  
> import Text.ParserCombinators.Parsec.Number (fractional, nat, sign)  
> import Text.Printf (PrintfType, printf)
```

And off we go.

The first thing we want is to model the data somehow. It'd be much more ideal to use `Data.Geo.Coordinate`, however, it uses a slightly different algorithm for computing the distance, and the difference in roundoff error is too great.

That said, we just model it directly as given:

```

> data Node = Node
>   { _number    :: Int
>     , _latitude :: Double
>     , _longitude :: Double
>   } deriving (Eq, Ord, Show, Typeable)

```

And of course generate some lenses so we don't end up doing too much work ourselves.

```

> makeLenses ''Node

```

Now we need some way to parse the file. Since we don't have to concern ourselves with the entire format at this time, we do the most simplistic approach possible.

First we ignore everything up until NODE_COORD_SECTION, then parse a whole bunch of nodes until EOF.

```

> parseNodes :: Parser [Node]
> parseNodes = do
>   manyTill anyChar $ try $ string "NODE_COORD_SECTION"
>   manyTill parseNode $ try $ string "EOF"

```

The format for the nodes is: <integer> <real> <real>.

We create a parser for this schema.

```

> parseNode :: Parser Node
> parseNode =
>   Node <$> (spaces *> nat)
>           <*> (spaces *> (sign <*> fractional))
>           <*> (spaces *> (sign <*> fractional))
>           <*> endOfLine

```

The algorithm given in the TSPLIB pdf. This is basically a direct copy. Any other implementation ends up with slightly different values.

```

> distance :: Node -> Node -> Int
> distance i j = floor (rrr' * acos (0.5 * ((1 + q1) * q2 - (1 - q1) * q3)) + 1)
>   where
>     rrr', q1, q2, q3, latI, latJ, longI, longJ, pi' :: Double
>     rrr' = _Sphere # rrr
>     q1 = cos $ longI - longJ
>     q2 = cos $ latI - latJ
>     q3 = cos $ latI + latJ
>     latI = radians i latDegrees latMinutes

```

```

> latJ = radians j latDegrees latMinutes
> longI = radians i longDegrees longMinutes
> longJ = radians j longDegrees longMinutes
> radians node d m = pi' * (node^.d + 5 * node^.m / 3) / 180
> latDegrees = latitude.to (fromInteger . round)
> latMinutes = latitude.to ((-) <*> fromInteger . round)
> longDegrees = longitude.to (fromInteger . round)
> longMinutes = longitude.to ((-) <*> fromInteger . round)
> pi' = 3.141592

```

The given radius of The Earth. Different values here also give different answers.

```

> rrr :: Sphere
> rrr = (6378.388 :: Double)^._Sphere

```

We construct a tour by pairing up each node with its immediate successor, and wrapping the end of the list around.

```

> tour :: [Node] -> [(Node, Node)]
> tour = zip <*> (uncurry (++) . swap . splitAt 1)

> tourDistance :: [(Node, Node)] -> Int
> tourDistance = sum . fmap (uncurry distance)

```

We provide some output formatting

```

> format :: PrintfType a => Int -> a
> format = printf "The distance of the tour in order 1-2-...-22-1 is: %d km\n"

```

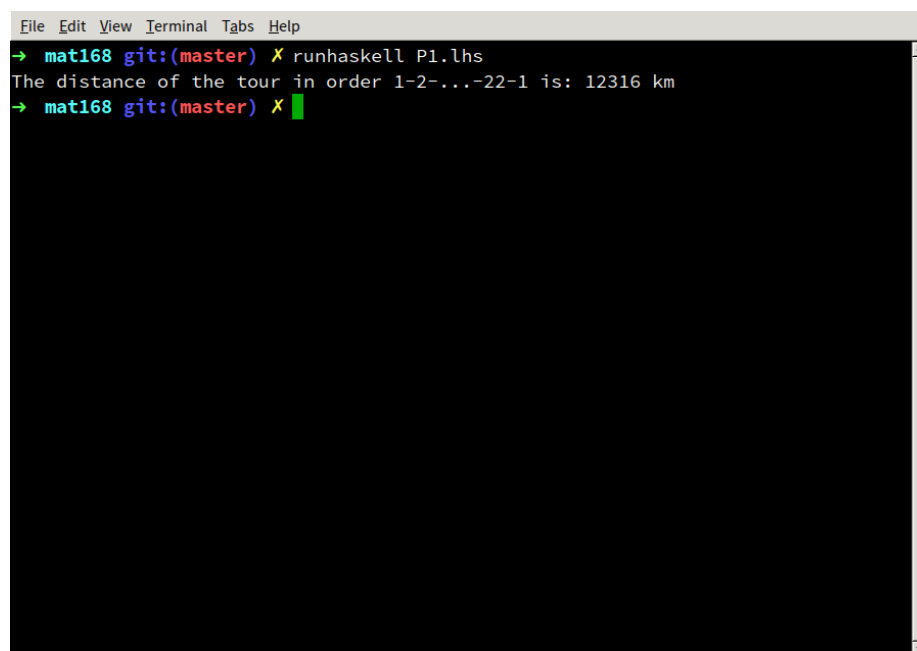
Finally, we actually attempt to run this program. First parse the file. If it doesn't parse, then print out the message and be done. If it parses fine, then create a tour, find the distance, and print it.

```

> main :: IO ()
> main = do
>   nodes <- parseFromFile parseNodes "ulysses22.tsp"
>   either print (format . tourDistance . tour) nodes

```

We can see this in action by running on the terminal:



```
File Edit View Terminal Tabs Help
→ mat168 git:(master) X runhaskell P1.lhs
The distance of the tour in order 1-2-...-22-1 is: 12316 km
→ mat168 git:(master) X
```

Figure 1: runit