COMP105: Programming Paradigms
Week 7 Homework Sheet

This is the homework sheet for **Week 7**. Complete your answers in a file named `week7.hs` and submit them to the "Week 7" assessment in SAM here

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Submission of the weekly homework sheets contributes 10% of the overall module mark, and each homework sheet counts equally towards this. Each homework sheet will be marked on a pass/fail basis. You will receive full marks for submitting a *reasonable attempt* at the homework. If no submission is made, or if a non-reasonable attempt is submitted, then no marks will be awarded.

The deadline for submission is

Friday Week 7 (27/11/2020) at 16:00.

Late submission is **not** possible. Individual feedback will not be given, but full solutions will be posted promptly after the deadline has passed.

If you feel that you are struggling with the homework, or if you have any other questions, then you can contact the lecturer at any point during the week via email, or you can drop in to the weekly Q&A session on MS Teams on Friday between 1PM and 4PM.

Lecture 19 - Custom types. Copy the following type declaration into your file.

```
data Direction = North | East | South | West deriving (Show)
```

After you load the code into ghci, you can use the new type. For example:

```
ghci> North
ghci> :t South
```

1. Modify the type definition so that the type also derives `Eq`. After reloading, test that your new definition works by running `North == North` and `North /= South`.

2. Modify the type definition so that the type also derives `Read`. Test that your new definition works by running `read "North" :: Direction`

3. Modify the type definition so that the type also derives `Ord`. test that your new definition works by running `North < East` and `max South West`. Do you understand why these queries return the results that they do?

4. Write a function `is_north :: Direction -> Bool` that returns `True` if the argument is `North` and `False` otherwise.

5. Write a function `dir_to_int :: Direction -> Int` that returns `1` if the argument is `North`, `2` if the argument is `East`, `3` if the argument is `South`, and `4` if the argument is `West`.

Lecture 19 - Types with data.

Copy the following type declaration into your file.

```
data Point = Point Int Int deriving (Show)
```

You can now build instances of point like so:

```
ghci> Point 2 4
Point 2 4
```

1. Write a function `same :: Int -> Point` that takes an integer `x` and returns `Point x x`.

2. Write a function `is_zero :: Point -> Bool` that returns `True` if the input is `Point 0 0` and `False` otherwise. When you call your function, make sure that you use brackets around the argument, like so: `is_zero (Point 0 0)`.

3. Write a function `mult_point :: Point -> Int` that takes `Point x y` and returns `x * y`.

4. Write a function `up_two :: Point -> Point` that takes `Point x y` and returns `Point x (y + 2)`.

5. Write a function `add_points :: Point -> Point -> Point` that adds two points together, so if the inputs are `Point a b` and `Point c d` then the output should be `Point (a+c) (b+d)`.

Lecture 21 - Maybe.

1. Recall the `Maybe a` type from the lectures. Try it out by typing the following into ghci.

   ```
   ghci> Just "hello"
   ghci> Just False
   ghci> Just 3
   ghci> Nothing
   ```

   Use the `:t` command to inspect the types of each of these values.

2. Write a function `not_nothing :: Eq a => Maybe a -> Bool` that returns `False` if the input `Nothing` and `True` otherwise. Note that the `Eq a` constraint is necessary if you intend to do something like `input == Nothing`, because `Maybe a` is only in `Eq` if `a` is also in `Eq`. Equality tests can be avoided by using pattern matching.

3. Write a function `mult_maybe :: Maybe Int -> Maybe Int -> Maybe Int` that returns `Just (x*y)` if the inputs are `Just x` and `Just y`, and returns `Nothing` if one or more of the inputs is `Nothing`.

Lecture 21 - Either.

1. Recall the `Either a b` type from the lectures. Try it out by typing the following into ghci.

   ```
   ghci> Left 'a'
   ghci> Left False
   ghci> Right "hello"
   ```

   Again, you can use the `:t` command to inspect the types of each of these values.

2. Write a function `return_two :: Int -> Either Bool Char` that takes one argument `n` and returns `Left True` if `n == 1` and returns `Right 'a'` otherwise.

3. Write a function `show_right :: Either String Int -> String` that returns `x` if the input is `Left x` and `show y` if the input is `Right y`.