This question deals with life and death. The following data type describes a sequence of actions in a chase:

```haskell
data ChaseActions = TigerMove ChaseActions
                   | LambMove ChaseActions
                   | LambDies | LambEscapes
```

(a) A tiger is feeling very hungry and a lamb is in the tiger’s view but not feeling particularly like being prey. When the tiger moves, it can move 3 meters in one stride. If the lamb moves, it can move 1 meter in one stride. Define a function `distance`, which given `ChaseActions` returns the distance between the tiger and the lamb. You may assume that the tiger and the lamb start off with a distance of 20 meters. Also show the type of the function. For example, `distance (TigerMove (LambMove (LambMove LambEscapes)))` should return 19, while `distance (TigerMove (TigerMove (LambMove LambEscapes)))` should return 15. For this function, you can assume the value returned can be negative.

[15 marks]

(b) Define a function `tigerStrides` which, given `ChaseActions`, returns the largest number of consecutive strides the tiger makes. Also show the type of the function. For example, `tigerStrides (TigerMove (LambMove (TigerMove LambEscapes)))` should return 1, while `tigerStrides (TigerMove (TigerMove (LambMove LambEscapes)))` should return 2.

[15 marks]
Define a function `correctKill` which, given `ChaseActions`, returns if the outcome of the chase is correct or incorrect. Thus, given a number of actions, the function will return true if the chase results truly in the lamb dying or the lamb escaping. A kill is one which results in the tiger reaching the lamb with distance 0, while an escape is one where the distance between them grows to larger than 30 meters. If the actions resolve to the race not being over or that the lamb did not really die or escapes, the function should return false. Also give the type of the function.

[20 marks]

Make `ChaseActions` an instance of the `Show` class where the `show` function should translate a sequence of tiger and lamb actions into a pictorial representation, showing in each line one step of the race. You should use `>` to represent the tiger, `*` to represent the lamb and `-` to represent a metre of distance. The first line should always represent the start of the race, with the tiger and lamb separated by 20 meters. For example, `show (LambMove (TigerMove (TigerMove LambDies)))` should return

```
>-------------------*
>-------------------*
>-------------------*
>-------------------*
```

[10 marks]

2. (a) With reference to your assignment, define the data type `Bitmap` and `BlobMap` that represents a bitmap and a series of blobs respectively.

[10 marks]

(b) Define the following function:

```haskell
type XPos = Int
type YPos = Int
type Color = Int

neighbourhoodColors :: Bitmap ->
  (XPos, YPos) -> [(XPos, YPos, Color)]
```

where `neighbourhoodColors` gives the position and color of pixels surrounding a particular x and y position. So for example if given position (5,5), the function will return the colors in the bitmap at position (4,4), (5,4), (6,4), (4,5), (6,5), (4,6), (5,6) and (6,6). Note that for simplicity you can assume that the input x and y positions will never be on the bitmap’s borders.

[15 marks]
(c) Define the following function

\[
\text{type Color} = \text{Int} \\
\text{paintUsed :: BlobMap} \rightarrow [\text{(Color, Double)}]
\]

where \text{paintUsed} will take a particular \text{BlobMap} and will give you back the amount of paint used for each color in the \text{BlobMap}. Note that the list returned should have only one entry per color. The returned result can be fractional. For reference the area of a circle is \( \pi \times \text{radius} \times \text{radius} \).

[15 marks]

3. (a) Define a data type \text{Dice} that represents a six sided dice using enumerated data types. Make this data type derive the \text{Eq} type class.

[5 marks]

(b) Make \text{Dice} an instance of the class type \text{Ord a} where

\[
\begin{align*}
\text{class} \ (\text{Eq a}) & \Rightarrow \text{Ord a} \ \text{where} \\
(\langle) & :: \ a \rightarrow a \rightarrow \text{Bool} \\
(\rangle=) & :: \ a \rightarrow a \rightarrow \text{Bool} \\
(\rangle) & :: \ a \rightarrow a \rightarrow \text{Bool} \\
(\langle=) & :: \ a \rightarrow a \rightarrow \text{Bool}
\end{align*}
\]

Note you cannot make use of the keyword \text{deriving} for class \text{Ord}.

[5 marks]

(c) Write a quickCheck property that guarantees that every number that is multiplied by itself is always a positive number.

[10 marks]

(d) For the function:

\[
\text{safeHead :: [a] -> Maybe a}
\]

define a quickCheck property that guarantees that when \text{safeHead} is called to a non empty list, the result is always the same as calling \text{Just head} on the list.

[10 marks]

(e) The following property sometimes returns a pass from quickCheck.

\[
\begin{align*}
\ -- \ \text{reverse reverses the list given to it} \\
\ -- \ \text{reverse:: [a] -> [a]}
\end{align*}
\]

\[
\text{prop\_RevID xs = reverse xs == xs}
\]

Can you explain why quickCheck passes the above property. Also show how you can make quickCheck always fail the above property.

[10 marks]