1. (a) If we have the following definitions:

\[ T \equiv \lambda xy.x \equiv K \]
\[ F \equiv \lambda xy.y \equiv KI \]
\[ \text{and} \equiv \lambda xy.xyx \]
\[ \text{if} \equiv \lambda pca.pca \]

show that

\[ \text{if } (\text{and } T \ F) \ (M) \ (N) \equiv N \]

Make sure you show all your derivation steps. [10 marks]

(b) Explain the meaning of referential transparency in Haskell and why is this important to any programming language. [10 marks]

(c) Given the following definitions:

```haskell```
```
data Word_Type = Noun
| Verb
| Adjective
    deriving (Eq,Show)
```
```
-- a dictionary containing a word, the type
-- of the word and the
-- explanation of that word
type Dictionary = [(String, Word_Type, String)]

-- given a dictionary and a type will give you
-- back only the words of that type
filterDictionaryByType :: Dictionary -> Word_Type
                     -> Dictionary

-- given the dictionary will look up a particular word
lookupWord::Dictionary -> String
              -> Maybe (Word_Type,String)

-- a phrase is ok if
-- it is a Noun Verb Noun
-- or it is a Noun Verb Adjective Noun
-- following function will return true
-- if the list of strings given
-- abides to these rules ex [noun,verb,noun] returns true
-- assume space seperate words
validPhrase:: Dictionary -> [String] -> Bool

Define the filterDictionaryByType function. [10 marks]

Define the lookupWord function. [10 marks]

Define the validPhrase function. [10 marks]
2. With reference to the assignment given in the year and the following definitions:

```haskell
type Point = (Float, Float)
type Position = (Float, Float)
type Angle = Float

-- the state of the turtle
type TurtleState = (Position, Angle)

-- the commands of the logo language
data Logo =
  Forward Float |
  Turn Float |
  Logo :> Logo |
  DoNothing
  deriving (Show)

-- given a list of logo program will
-- generate a single logo program that is the
-- concatenation of all elements in the
-- list
seqLogo :: [Logo] -> Logo

-- given a logo program and a turtle state,
-- this function will output a list of strings
-- containing all the commands
-- and the new turtle state
play :: Logo -> TurtleState -> ([String], TurtleState)
```

(a) Define the `seqLogo` function.  

(b) Define the `play` function. 

Please note that there are some minor differences between the assignment and this definition.
3. With reference to the code snippet below:

```haskell
type Mark = Int
type QuestionText = String

data Difficulty = Easy | Medium | Hard
    deriving (Show)

-- an exam is made up of sections
-- with each section having more sections
-- or a question
-- a question is made up of multiple parts also
data Exam a =
    Sections [Exam a]
  | Questions [a]
```

(a) Define the following function that takes an exam as input and will return the difficulty of the most difficult question part.

```haskell
max_difficulty :: Exam Difficulty -> Difficulty
```

[5 marks]

(b) Define the follow function that will give the total marks of all questions in the exam.

```haskell
addupMarks :: Exam Mark -> Mark
```

[10 marks]

(c) Make Exam QuestionText an instance of Show.

[10 marks]

(d) Define from scratch the following function that given an exam of one format will change it to another format.

```haskell
examText :: Exam (QuestionText, Difficulty, Mark) -> Exam QuestionText
```

[10 marks]

(e) Define the following higher order function that applies the first argument to the second argument to perform a transformation.

```haskell
mapExam :: (a -> b) -> Exam a -> Exam b
```

[10 marks]

(f) Now make use of the function in (e) to redefine examText from (d).

[5 marks]