CS5502: Functional Programming(End Semester exam)

Instructions

1. Please type or write the answer legibly.
2. Send the answer book via email with the subject line `[functional programming] End Sem Exam`. The answer book itself should be a pdf attachment (ensure that the size is not too big). I will acknowledge the receipt with a reply to the mail.
3. Your (possibly multi-page) answer book should be attached to the above mail as a single pdf file. An easy method to do this is to write answers in a4 sized pages and scan (and compress) each of them separately. Use a tool like pdftk to make a multi-page pdf out of it.
4. Write your name, roll number and signature on the first page of the answer book. Please number the pages and also sign on the top of each of them.
5. In case you are sending a typed answer script, you still need to attached a printed/scanned copy with your signatures on it.

Question 1. (5 points) The `sequence :: Monad m => [m a] -> m [a]` runs a list of monadic actions and returns the list of its results. Give its definition using the *`do`-notation*.

Question 2. (5 points) Rewrite the function `foo` given below using the `Applicative` operators `<$>` and `<*>`.

```
foo act = do x  <- act
             xs <- act
             pure (x + xs)
```

Notice that the type of the function would change in the process. Can you explain why?

Question 3. (10 points) In the class, we defined the `Functor` and `Monad` instance for the state monad defined as follows

```
newtype State s a = State { runState :: s -> (a,s) }
```

Give its `Applicative` instance.

Question 4. IIT Palakkad recently bought a vending machine capable of vending two snacks, *Samosa* and *Vada* which costs Rs 8 and Rs 7 respectively. At any point of time the user might (1) Drop some money (2) Demand one of the snack (3) Demand the change. The vending machines action can be controlled by performing the following `IO` actions which is given to you as a Haskell library. Your controller program can call this whenever it wishes.

```
data Snack = Samosa | Vada
-- The user input
data Input = Money Int | Demand Snack | Change
```

```
-- This action dispenses a given snack to the user.
snack  :: Snack  -- the snack to be given out.
       -> IO ()

-- This action pays a given amount of money to the user. Can be used
-- to handle the change action.
pay :: Int -- Give out so-much rupee.
    -> IO ()

-- Wait for the next user input and return it.
getUserInput :: IO Input

-- Use to display some message to the user. Info/Error etc
display :: String     -- The message to be displayed
        -> IO ()
```

The task is to write a program that manages the vending machine, accounting the money paid by the user and giving the necessary change. As an example, suppose I drop 2 rs and demand for a vada the program should refuse but if I drop Rs 5, Rs 5 and demand a vada it should give out. Subsequently, if I demand change it should give out 3rs (10 - 7 for the vada).

(a) (1 point) Using 'StateT' define the `VendM` monad can that keep track of the money deposited by the user as well as perform IO actions required for vending.

```
type VendM a = ....
```

(b) (3 points) Define the function `change :: VendM ()` which will return the remaining amount to the user.

(c) (3 points) Define the function `serve :: Snack -> VendM ()` that checks the current balance and serves the given snack if the user has already deposited enough money.

(d) (6 points) Define the infinite loop `vend :: VendM ()` which takes care of waiting for user action and performing the necessary action.

(e) (2 points) Define the `main :: IO ()` of your program that essentially calls the `vend` action with the starting balance of `0` Rs.

You may write other helper function to clean up the code.