Introduction to Functional Programming

Assignment 5-6: Higher-Order Functions and Functors

Problem Description:

Students need to implement **Student** data type that holds the basic information about a student, including their grades, attendance, and personal information. Then, they need to write several functions that will allow you to manipulate list of Student objects using **higher-order functions** and **functors**.

Student Data Type

```
data Student = Student {
  firstName  :: String,   -- First name of the student
  lastName   :: String,   -- Last name of the student
  grades     :: [Int],    -- List of grades (on a scale from 0 to 100)
  attendance :: Float     -- Attendance percentage
}  deriving (Show)
```

Required Functions

You need to implement the following functions based on the **Student** data type. These functions should utilize **higher-order functions** (functions that take other functions as arguments or return them as results) and **functors.**

1. calculateGPA

This function calculates the **GPA** of a student based on their grades. The GPA is calculated using the following mapping from grades to GPA:

Grade	GPA
90-100	4.0
85-89	3.5
80-84	3.0
75-79	2.5
70-74	2.0
65-69	1.5
60-64	1.0
Below 60	0.0

Signature:

```
calculateGPA :: Student -> Float
```

2. filterStudents

This is a **generic filter function** that will take a predicate (a function that returns a boolean) and filter a list of students based on that predicate. The predicate can be based on attendance, GPA, or any other property of the student.

Signature:

filterStudents :: (Student -> Bool) -> [Student] -> [Student]

Predicate Functions for filterStudents

You need to implement the following **predicate functions** that can be used with filterStudents to filter students based on different criteria:

2.1 Filter by GPA:

This predicate will check if a student's GPA is greater than or equal to a given value.

Signature:

filterByGPA :: Float -> Student -> Bool

Example usage:

filterStudents (filterByGPA 3.0) students

2.2 Filter by Attendance:

This predicate will check if a student's attendance is greater than or equal to a given percentage.

Signature:

filterByAttendance :: Float -> Student -> Bool

Example usage:

filterStudents (filterByAttendance 80) students

3. fmapGrades

This function will demonstrate how to use **functors**. Specifically, it will apply a transformation (a function) to the student's grades (i.e. curving the grades).

Signature:

fmapGrades :: (Int -> Int) -> Student -> Student

Description: This function takes a transformation function for the grades (such as increasing each grade by a certain value) and applies it to the grades of the student.

4. fmapAttendance

This function will also demonstrate **functors** but for the student's attendance. It will apply a transformation to the student's attendance percentage.

Signature:

fmapAttendance :: (Float -> Float) -> Student -> Student

Description: This function takes a transformation function for attendance (such as increasing attendance by a certain percentage) and applies it to the student's attendance.

Examples of How to Use These Functions

Here are some example usages for the functions you've implemented:

1. **Calculate GPA**:

   calculateGPA (Student "Alice" "Smith" [90, 85, 78] 95)

   -- Result: 3.0

2. **Filter by GPA**:

   filterStudents (filterByGPA 3.0) students

   -- Filters students with GPA >= 3.0

3. **Filter by Attendance**:

   filterStudents (filterByAttendance 80) students

   -- Filters students with attendance >= 80%

4. **Apply transformation to grades**:

   fmapGrades (+5) (Student "Bob" "Lee" [75, 82, 90] 85)

   -- Result: Student {firstName = "Bob", lastName = "Lee", grades = [80, 87, 95], attendance = 85}

5. **Apply transformation to attendance**:

   fmapAttendance (+5) (Student "Charlie" "Brown" [80, 90, 88] 70)

   -- Result: Student {firstName = "Charlie", lastName = "Brown", grades = [80, 90, 88], attendance = 75}

Assignment requirements:

1. Do not submit the solution of a given assignment
2. Students will be required to do live coding in class by a given new additional task (add new functions, change existing ones or solve a similar problem)
3. **Submit the code after you complete in-class tasks**