(20/80 marks)

Question 1: Basics.

Question 1.a. (10 marks)

Provide a definition of a function called `translate` that translates a given 3d point in a given direction. The first argument of the function is the point and the second argument is the direction. The *x*, *y*, and *z* coordinates of both parameters are arbitrary precision integers. You may carry out the translation by pairwise adding the *x*, the *y*, and the *z* coordinates. E.g. `translate (1,2,0) (3,7,0)` results in `(4,9,0)`.

Briefly explain how you obtained the *x*, *y*, and *z* coordinates of the arguments of the function. How does `Haskell` refer to this technique?

Question 1.b. (10 marks)

This question is about an implementation of a user-defined (as opposed to `Haskell`) list.

- A list can be empty or non-empty.
- An empty list stores no data elements.
- A non-empty list consists of a head and a tail element: the head is a character and the tail is a list.

Using record syntax, provide a data definition for the user-defined list. Provide two good reasons why your definition is better than a definition that doesn't use record syntax.

(20/80 marks)

Question 2: List Processing.

Question 2.a. (4 marks)

Define a function that takes two arguments and returns a list consisting of these arguments: the first argument should be the first member of the resulting list. You get one bonus mark for a correct answer that doesn't use a comma.

Question 2.b. (8 marks)

For this question you must provide two implementations of a function that takes in a list consisting of lists and returns a list consisting of the lengths of the elements in the function's argument. The *n*th number in the output should be the length of the *n*th element in the function's argument.

1. The first implementation should use a list comprehension.
2. The second implementation should use recursion.

Question 2.c. (8 marks)

Provide a definition of a function that takes in a list of 2-ary tuples and returns the sum of the members of the tuples. E.g. the function should return 5 for the argument `[(1,2),(2,0)]` and 0 for the argument `[(1,-1)]`.

Question 3: Higher-Order Functions and Advanced Expressions. (20/80 marks)

Question 3.a. *(3 marks)*
Provide the type of the function `map`. Is the function `map` a higher-order function? Explain your answer.

Question 3.b. *(1 mark)*
What is a partial application? There is no need to explain your answer.

Question 3.c. *(3 marks)*
Provide an example of a partial application, state the type of the partial application, and provide a description of the semantics of the partial application.

Question 3.d. *(3 marks)*
What is an anonymous function (λ-expression)? Provide an example of an anonymous function that adds two numbers.

Question 3.e. *(4 marks)*
Provide the type of the function `.` (dot), state the purpose of the function, and provide an example that shows how to use the function.

Question 3.f. *(6 marks)*
The Fibonacci numbers are given by 0, 1, 1, 2, 3, ... Given two consecutive numbers, a and b, in the sequence you can compute the next number in the sequence by computing $a + b$. Provide a self-referential definition for an "infinite" list consisting of the Fibonacci numbers. *Hint: use* `zipWith` *and* `+`.

Question 4: Types and Type Classes. (20/80 marks)

Question 4.a. *(14 marks)*
Provide a user-defined, polymorphic class for two-dimensional coordinates. The name of the class should be `Coordinate`. The class should define:

- A function called `createCoordinate` for creating an instance of the class;
- A function called `getFirst` for getting the first coordinate of an instance of the class;
- A function called `getSecond` for getting the second coordinate of an instance of the class; and
- A function called `addCoordinates` for *adding* two instances of the class; Here two instances are added by pairwise adding their corresponding coordinates.

Question 4.b. *(6 marks)*
Provide an implementation of the `Coordinate` class from the previous question for instances of `Haskell`'s built-in pair class.