Problem Sheet 1.1: Introduction to Haskell

Function Types

In the Haskell file, you have two functions, `isEven` and `square`. Both functions take integers as inputs, but output different types. Both the type and definition of `isEven` has been given to you.

```
isEven :: Int -> Bool
isEven n = if (n `mod` 2 == 0)
           then True
           else False
```

a) Give the function `square` a type signature. Compile your .hs file to make sure type checking runs successfully.

b) The function `squareEven1` should square even numbers, while returning the input number for odd numbers. Define `squareEven1` , without using the functions `square` or `isEven` .

```
*Main> squareEven1 5
5
*Main> squareEven1 4
16
```

c) Often, we will use previously defined functions to build up more complex function. Define `squareEven2` using both `square` and `isEven` .

d) Think about the expression ``(n `mod` 2 == 0)`` . What is the type of this expression? After reflection, try to rewrite the `isEven` function without the use of `if...then ...else` .

Guards

In Haskell, it is better to use guard syntax rather than `if...then ...else` . As an example, we define the function `odd` using guard syntax. Below `odd` is an incomplete function `grade` which takes an integer to a string that outputs the grade that should be given to marks between 0 and 100.

a) “Medium Pass” should be awarded to marks between 50 and 59. Adapt the definition of `grade` to reflect this.

b) Add cases for “High Pass” for marks between 60 and 69, “Merit” for marks between 70 and 79, and “Distinction” for marks between 80 and 100.

c) Marks above 100 are invalid. Use `otherwise` to write a case for these marks.

```
*Main> grade 65
"High Pass"
*Main> grade 80
"Distinction"
*Main> grade 120
"Invalid Mark"
```

Recursion

You've already seen the recursive function `factorial`, and we have put it in the top of this section in the Haskell file. The `factorial` applied to `n` multiplies all numbers from `1` to `n`.

a) The nth triangle number is defined to be `1 + 2 + ...+ (n-1) + n`. In a similar way to `factorial`, define the function `triangle` to calculate the nth triangle number for input `n`.

```
*Main> triangle 10
55
```

b) Below `factorial`, we give you the function `total`, which sums up the elements of a list of integers. Define a similar function `multiple` which multiplies together all the elements of the list.

```
*Main> multiple [2,6,4]
48
```

c) We can write the list of numbers `1` to `n` as `[1..n]`. Using this, we can define a function `triangle` that computes the same outputs as `triangle`. In a similar way, define the function `factorial` using `multiple`.

d) The Euclidean algorithm for the greatest common divisor (GCD) of two natural numbers is this: for input x and y, if x and y are equal, that is also their GCD; otherwise, take the GCD of the smaller one of x and y and the difference between x and y. Implement this as the function `euclid`.

e) `Main> euclid 45 25`
5
f) `Main> euclid 49 91`
7

Multiple arguments and Partial Application

As you have seen, functions of more than one variables can be **partially applied** to create functions of fewer variables. We give an example, `gcd`, which calculutes the greatest common divisor of the input and 5.

a) We have defined the function `facDiv`, which, given two integers, returns the factorial of the larger integer divided by the smaller integer. Give the type signature for and implement the function `facDiv7`, which given an integer applies `facDiv` to that integer and 7.

```
*Main> facDiv7 9
72
```

b) The function `facTri` takes as input an integer `n` and a Boolean, and returns the factorial of `n` if the Boolean is `True`, and the nth Triangle number if the Boolean is false. Write the function `facEvenTriOdd`, which takes an integer `n` as input, giving the factorial of `n` if `n` is even, and the nth triangle number if `n` is odd. You can use your `isEven` function defined above. Is it possible to write this function using partial application? If not, why not? You can discuss your ideas in the forums.