Advanced Functional Programming: Assignment 1 (Wed, 03/20/2019)
Topic: Streams, Generators, Selectors, and Combinations thereof
Submission deadline: Wed, 04/03/2019 (3pm) (two weeks!)

Regarding the deadline for the second submission: Please, refer to „Hinweise zu Organisation und Ablauf der Übung“ available at the homepage of the course.

Store all functions to be written for this assignment in a top-level file assignment1.hs of your group directory. Comment your program meaningfully; use auxiliary functions and constants, where reasonable.

Important: Do not use self-defined modules! If you want to re-use functions (written for earlier assignments), copy these functions to the new submission file. An import declaration for self-defined modules will fail, since only the submission file assignmenti.hs, where \( 1 \leq i \leq 8 \) (tentatively), denotes the running number of the assignment, will be copied for the (semi-automatical) evaluation. No other file in addition to assignmenti.hs will be copied.

1. Implement the generator and selectors:
   - repeat (generator)
   - within (selector)
   - relative (selector)

   of Chapter 1.3 as type-general as possible in Haskell, and test different combinations of them including the examples for approximately
   - computing the square roots of positive real numbers
   - integrating continuous 1-ary real functions
   - differentiating continuous 1-ary real functions.

   To this end hide the name repeat defined in the standard prelude using the hiding clause and additionally implement the functions:
   - next
   - easyintegrate
   - integrate (1st version of integrate of Chap. 1.3)
   - integrate_eff (Improved 2nd version of integrate of Chap. 1.3)
   - easydiff
   - differentiate

   together with the auxiliary functions they refer to and the generator/selector combinations:
   - sqrt :: InitialApprox -> Epsilon -> SquareArg -> Approx
   - relativesqrt :: InitialApprox -> Epsilon -> SquareArg -> Approx
- \texttt{intgrt :: Map} \rightarrow \texttt{Low} \rightarrow \texttt{High} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Area}
  (Analogue to the generator/selector combination \texttt{sqrt})

- \texttt{relativeintgrt :: Map} \rightarrow \texttt{Low} \rightarrow \texttt{High} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Area}
  (Analogue to \texttt{relativesqrt})

- \texttt{intgrteff :: Map} \rightarrow \texttt{Low} \rightarrow \texttt{High} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Area}
  (Improved, more efficient variant of \texttt{intgrt})

- \texttt{relativeintgrteff :: Map} \rightarrow \texttt{Low} \rightarrow \texttt{High} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Area}
  (Improved, more efficient variant of \texttt{relativeintgrt})

- \texttt{diff :: Map} \rightarrow \texttt{XCoordinate} \rightarrow \texttt{InitialH} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Slope}
  (Analogon zu \texttt{sqrt})

- \texttt{relativediff :: Map} \rightarrow \texttt{XCoordinate} \rightarrow \texttt{InitialH} \rightarrow \texttt{Epsilon} \rightarrow \texttt{Slope}
  (Analogue to \texttt{relativesqrt})

where:

- \texttt{type InitialApprox = Double} \quad \text{-- Only values > 0}
- \texttt{type Epsilon = Double} \quad \text{-- Only values > 0}
- \texttt{type SquareArg = Double} \quad \text{-- Only values > 0}
- \texttt{type Approx = Double} \quad \text{-- Only values > 0}
- \texttt{type Map = Double \rightarrow Double}
- \texttt{type Low = Double} \quad \text{-- Lower interval bound}
- \texttt{type High = Double} \quad \text{-- Upper interval bound}
- \texttt{type Area = Double}
- \texttt{type XCoordinate = Double}
- \texttt{type InitialH = Double} \quad \text{-- Only values > 0}
- \texttt{type Slope = Double}

Use the standard type \([\ ]\) for both lists and streams, and the type \texttt{Double} as the implementation of the real numbers. All functions yield the value of the most recently computed approximation, i.e., the most precise approximation computed when the computation is stopped.

2. \textbf{Without submission}: The functions \texttt{integrate}, \texttt{integrate\_eff}, and \texttt{differentiate} are generators themselves. Unlike \texttt{differentiate}, however, \texttt{integrate} and \texttt{integrate\_eff} do not make use of the generator \texttt{repeat}.

   How could a generator \texttt{repeat2} look like allowing to implement \texttt{integrate} and \texttt{integrate\_eff} analogously to \texttt{differentiate} (which makes use of \texttt{repeat}), and being reusable for other tasks in the same way as \texttt{repeat} is?

3. Consider the sequence(s) \((x_i)_{i \in \mathbb{N}_0}\) of real numbers, whose elements are computed according to the rule (for \(n \geq 0\)):

   \[ x_{n+1} = ax_n(1-x_n) \]

   where \(a\) is a real valued constant and \(x_0\) a real valued initial value with \(0 \leq a \leq 4\) and \(0 \leq x_0 \leq 1\).
Write a Haskell function `next2` over the type synonyms:

```haskell
 type Value_a = Double  -- 0 <= a <= 4
 type Value_x0 = Double  -- 0 <= x0 <= 1
 type Value_xn = Double
 type Value_xnplus1 = Double

next2 :: Value_a -> Value_xn -> Value_xnplus1
```

and by means of the generators and selectors `repeat`, `within`, and `relative` of part 1 the generator/selector combinations:

- `sequence`
- `relativesequence`

analogously to the generator/selector combinations `sqrt` and `relativesqrt`.

4. **Without submission**: Investigate the behavior of convergence of `sequence` and `relativesequence` in dependence of the value of `a`. To this end, choose different values of `a` from the intervals:

- `0 ≤ a < 1`
- `1 ≤ a < 3`
- `3 ≤ a ≤ 3.449`
- `3.449 < a ≤ 4`

Combine the generator (expressions) also with selectors like `take n` for increasing values of `n ∈ \mathbb{N}`, and derive a hypothesis about the behavior of the elements of the sequence in dependence of the selected value `a` from that. Supposed your hypothesis is valid, are the selectors `within` and `relative` meaningful for all values of `a`?

5. Let `f : \mathbb{IR} \to \mathbb{IR}` be a continuous real function. Function `f` has a change of sign (Vorzeichenwechsel) in the interval `I = [a, b] ⊆ \mathbb{IR}`, if there is a subinterval `I_0 = [a_0, b_0] ⊆ I` with

\[ f(a_0) f(b_0) < 0 \]

According to the intermediate value theorem (Zwischenwertsatz) for continuous real functions there is at least one root (Nullstelle) of `f` in the interval `I_0 = [a_0, b_0]`, i.e., there is `x \in \mathbb{IR}` with `a_0 ≤ x ≤ b_0` and `f(x) = 0`.

Using an interval nesting approach (Intervallschachtelungsverfahren), we can approximate such a root as follows:

Let `I_t = [a_t, b_t]` be an interval with `f(a_t) f(b_t) < 0`, and let `x_t = \frac{1}{2} (a_t + b_t)` be the centre (Mittelpunkt) of the interval `I_t`.

- If `f(x_t) = 0`, then `x_t` is a root of `f`, and the computation stops.
- If `f(x_t) ≠ 0` and `f(x_t) f(b_t) < 0`, then a new interval `I_{t+1} = [a_{t+1}, b_{t+1}]` is constructed according to the rule:

\[ a_{t+1} = x_t \quad \text{and} \quad b_{t+1} = b_t. \]
• If \( f(x_t) \neq 0 \), \( f(x_t)f(b_t) > 0 \) and \( f(a_t)f(x_t) < 0 \), then a new interval \( I_{t+1} = [a_{t+1}, b_{t+1}] \) is constructed according to the rule:

\[
a_{t+1} = a_t \quad \text{and} \quad b_{t+1} = x_t.
\]

Write a Haskell function \( \text{nextinterval} \) over the type synonyms:

```haskell
type Interval = (Double,Double)
type InitialInterval = Interval
type Map = Double -> Double -- Only continuous functions
type Epsilon = Double -- Only values > 0
nextinterval :: Map -> Interval -> Interval
```

and based thereon a generator:

```haskell
intervalenesting :: Map -> InitialInterval -> [Interval]
```

computing a stream of intervals following the above approach when applied to a continuous map \( f \) and an initial interval \( I \).

Combine the generator \( \text{intervalenesting} \) with two modified (possibly type-adjusted) selectors \( \text{within2} \) and \( \text{relative2} \) (whose meaning corresponds to that of the selectors \( \text{within} \) and \( \text{relative} \) of part 1) to two generator/selector combinations:

```haskell
null :: Map -> InitialInterval -> Epsilon -> Interval
relativenull :: Map -> InitialInterval -> Epsilon -> Interval
```

which stop the interval nesting approach, when the absolute value (Abolutbetrag) of the difference resp. the ratio of two successive intervals coincide or is lower than a predetermined \( \epsilon > 0 \). In both cases, the most recently computed interval is provided as the result, i.e., the most precise approximation computed when the computation is stopped.

**Important:**

• **Login data:** You should have received your login data for the computer g0.complang.tuwien.ac.at by 20 March 2019. The login data will have been sent by email to your generic mail address e<matrikelnummer>@student.tuwien.ac.at. Once received, please, log in as soon as possible on the computer g0 (e.g., via ssh) and set your initial password to a new one of your own.

• **Submitting assignments:** Your programs will be (semi-automatically) evaluated on the machine g0 using the Hugs interpreter. If you use a different tool (such as GHC) or computer for developing your programs, please, double-check well in time before the submission deadline that your programs behave also on the computer g0 using Hugs as intended and expected by you.