Introduction to Programming: Assignment 3

Due: December 5, 2021. 11.59 pm

Important Instructions:

- Submit your solution in a single file named `loginid.3.hs` on Moodle. For example, if I were to submit a solution, the file would be called `spsuresh.3.hs`.
- I have provided a `template.hs` file, where I have stated the function names and types. Please do not change them. You may define auxiliary functions in the same file, but these function names should not be changed, and you must provide the proper definition for each of these.
- Please remember to rename your file to `loginid.3.hs` before submitting.
- Deviation from the instructions will result in marks being reduced.
- If your Haskell file does not compile, or if your function names differ from the ones I have specified (ensure that you use the exact sequence of small letters and capital letters), you will receive no marks.

---

1. A group of `n` children labelled `[1..n]` are playing a counting out game with parameter `k`, to select a leader among themselves. The game proceeds as follows:

   - The children form a circle.
   - Child `1` is marked as an invalid candidate.
   - Ignoring the invalid children, the child who is `k` places (modulo the number of valid children) to the right of the last invalid child is again marked invalid.
   - The process is repeated until all but one child are marked invalid.
   - The last remaining child is selected as the leader

   Here are the steps of the process run with `5` children and `k=2`:

   ```
   1 2 3 4 5
   1 2 3 4 5
   1 2 3 4 5
   1 2 3 4 5
   1 2 3 4 5
   ```

Define `selectLeader :: Int → Int → Int` such that `selectleader n k` is the label of the leader selected when the process starts with `n` children and parameter `k`.

Sample cases:

```
selectLeader 100 5 = 43
selectLeader 5 2   = 2
selectLeader 10 5  = 9
```

2. Modify the above program to define `selectLeader' :: Int → Int → [[Int]]` which generates all the lists in order as you play the game.

   Sample cases:

```
selectLeader' 10 5 =   [[1,2,3,4,5,6,7,8,9,10],
                        [  2,3,4,5,6,7,8,9,10],
                        [  2,3,4,5,  7,8,9,10],
                        [    3,4,5,  7,8,9,10],
                        [    3,4,5,  7,  9,10],
                        [    3,4,    7,  9,10],
                        [    3,      7,  9,10],
                        [    3,          9,10],
                        [    3,          9   ],
                        [                9   ]]
selectLeader' 9 4  =   [[1,2,3,4,5,6,7,8,9],
                        [  2,3,4,5,6,7,8,9],
                        [  2,3,4,  6,7,8,9],
                        [  2,3,4,    7,8  ],
                        [  2,  4,    7,8  ],
                        [      4,    7,8  ],
                        [            7,8  ],
                        [            7    ]]
```

   I have formatted the output for clarity. You just need to produce a list of list of integers.

3. This problem is related to rational numbers and their continued fraction representation.

   Rational numbers are represented using the data type **`Rational`** in Haskell. The ratio p/q is represented using the **`(%)`** operator defined in **`Data.Ratio`**. Acquaint yourself with the functions **`numerator`** and **`denominator`** defined in **`Data.Ratio`**. (Note: **`numerator rat`** can be positive or negative, but **`denominator rat`** is always positive.) Also try out the function **`fromIntegral`** on various test cases in **`ghci`**.

A finite continued fraction is any expression of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots \cfrac{1}{a_n}}}$$

where a_0 is an integer and each a_i is a positive integer, for $i \geq 1$. This is succinctly represented as the list $[a_0; a_1, a_2, \ldots, a_n]$. (Note the semicolon after the first entry.) A finite continued fraction can be calculated to a rational of the form $\frac{p}{q}$. On the other hand, every rational number can be expressed as a finite continued fraction. For example, the rational number $\frac{42}{31}$ can be rendered as a continued fraction using the following steps:

$$\begin{aligned}
\frac{42}{31} &= 1 + \frac{11}{31} \\
&= 1 + \cfrac{1}{\frac{31}{11}} \\
&= 1 + \cfrac{1}{2 + \frac{9}{11}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{\frac{11}{9}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \frac{2}{9}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{\frac{9}{2}}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{4 + \frac{1}{2}}}}
\end{aligned}$$

Thus one continued fraction corresponding to $\frac{42}{31}$ is $[1; 2, 1, 4, 2]$. A continued fraction corresponding to $-\frac{26}{21}$ is $[-2; 1, 3, 5]$. The continued fraction representation is not unique. For instance, both $[0; 1, 1, 1, 1]$ and $[0; 1, 1, 2]$ represent $\frac{3}{5}$.

Define a function `computeRat :: [Integer] → Rational` that takes a *nonempty* list of integers, such that all but the first element are positive, and returns the rational number corresponding to it.

Define a function `cf :: Rational → [Integer]` that takes a rational number as input and returns a continued fraction corresponding to it.

Sample cases: (Since there are multiple answers possible for `cf`, the cases below are only indicative. We will check the correctness of your solution by actually computing the inverse and checking.)

```
cf (33%42)              = [0,1,3,1,2]
cf (22%7)               = [3,7]
cf (23%7)               = [3,3,2]
cf (26%21)              = [1,4,5]
cf (-26%21)             = [-2,1,3,5]
cf (-42%31)             = [-2,1,1,1,4,2]
computeRat [3,3,2]      = 23%7
computeRat [3,7]        = 22%7
computeRat [1,4,5]      = 26%21
computeRat [-1,1,1,1,1] = (-2)%5
```

4. Just like finite continued fractions represent rationals, infinite continued fractions of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots}}$$

correspond to irrational numbers. For example, $\sqrt{6}$ can be written as[1]

$$2 + \cfrac{1}{2 + \cfrac{1}{4 + \cfrac{1}{2 + \cfrac{1}{4 + \ldots}}}}$$

This is represented more succinctly as $[2; 2, 4, 2, 4, \ldots]$. If we truncate this list at some finite point, we get a finite rational approximation for $\sqrt{6}$.

Assume the following Haskell definitions:

```
root6 :: Double
root6 = sqrt 6
```

[1] This representation can be verified as follows: letting

$$y = \cfrac{1}{2 + \cfrac{1}{4 + y}},$$

we see that $y = \dfrac{4+y}{9+2y}$ and hence $(9+2y)y = 4+y$. In other words, $2y^2 + 8y - 4 = 0$, i.e. $y^2 + 4y - 2 = 0$. It can be checked that $\sqrt{6} - 2$ is a root of this equation. So $\sqrt{6} = 2 + y$.

```
evalRat :: Rational → Double
evalRat x = fromIntegral (numerator x) /
            fromIntegral (denominator x)
```

Write a function `approxRoot6 :: Double → Rational` which returns a close rational approximation for $\sqrt{6}$, i.e. it accepts an input `epsilon` and returns some `r :: Rational` such that `abs (root6 - evalRat r) < epsilon`.

Note: Do not use the **`approxRatio`** function from `Data.Ratio`.

Sample cases: (Since there are multiple answers possible, the cases below are only indicative. We will check the correctness of your solution by actually calculating if the error is less than `epsilon`.)

```
approxRoot6 0.0001          = 218 % 89
approxRoot6 0.000001        = 2158 % 881
approxRoot6 0.0000000000001 = 20721118 % 8459361
```

5. Given a finite list of integers $l = [x_1, x_2, \ldots, x_n]$, we want to compute the *maximum segment sum*, $mss(l)$, to be the maximum value in the set

$$\{x_i^3 + \cdots + x_j^3 \mid 1 \leqslant i \leqslant j \leqslant n\}.$$

Write a program `mss :: [Int] → Int` that computes the maximum segment sum of a given list. Bonus marks if the running time is $O(n)$.

Sample cases:

```
mss [4,10,3,12,-1,1,-1,-7,6,-13]                        = 2819
mss [4,6,1,4,0,4,5,-3,2,8,-5,2,2,-1,-2,0,9,2,10,2]      = 2654
mss [-6,11,7,18,-13,-13,11,11,4,8,-12,-4,-3,-15,20]     = 9156
mss [4,4,-4,4,-5,-4,-4,-4,-7,8,-4,2,2,-7,3,5,6,-1,2,-3] = 512
```

6. Given a sequence `xs :: Eq a ⇒ [a]`, a *longest palindromic subsequence* is a subsequence `ys` of `xs` such that `reverse ys = ys`. This is not necessarily unique, but the length of such a sequence is unique.

Define a function `lps :: Eq a ⇒ [a] → (Int, [a])` that returns the length of the longest palindromic subsequence, as well as a subsequence of that length.

Sample cases:

```
lps "gultnhebrpuuowjrtpfw"  = (6,"truurt")
lps "edgnqqavflgfpzubuiaq"  = (7,"qaflfaq")
```

```
lps "nndrvtgiorptiateaihs"  = (5,"tioit")
lps "quwzpirtwnrazkbjrzcc"  = (5,"zrtrz")
```