National Undergraduate Programme in Mathematical Sciences

National Graduate Programme in Computer Science

# Functional Programming in Haskell

Assignment 2

---

- **Due date**: September 30, 2023, 2359 hours
- Submit your solution in a single file named `loginid.hs` on Moodle. For example, if I were to submit a solution, the file would be called `spsuresh.hs`. You may define auxiliary functions in the same file, but the solutions should have the function names specified by the problems.
- Remember that function names should start with a lowercase letter, and there should be no indentation for non-local functions.
- Please compile and check that the sample cases are satisfied before submitting!
- I have provided a `template.hs` with the function names and types, and some useful helper functions. Replace the **`undefined`** in the file with the actual definition. You are free to define other helper functions in the same file. Copy/rename this file to `loginid.hs`, complete the definitions, and submit.

---

1. Define the following function that checks whether a list of integers is an arithmetic progression. We will assume that any list whose length is at most 2 is a trivial arithmetic progression.

```
isAP :: [Integer] → Bool
```

   Here are some sample cases:

```
isAP []                    = True
isAP [2]                   = True
isAP [-22, -385]           = True
isAP [2,19,36,53,70,87]    = True
isAP [2,19,36,52,69,86]    = False
```

2. Define a Haskell function

```
targetSum :: Integer → [Integer] → Bool
```

   such that `targetSum t l` evaluates to **`True`** exactly when there exist two distinct indices `i, j` such that `l !! i + l !! j == t`. Note that the indices must be distinct, even if the elements in those positions are equal.

   Here are some sample cases:

```
targetSum 100 []                    = False
targetSum 100 [50]                  = False
targetSum 10 [4,6]                  = True
targetSum 10 [4,5,5,6]              = True
targetSum 20 [1,2,3,4,5,6,7,8,9,10] = False
targetSum 20 [10,9,8,6,7,8,9,10]    = True
```

3. The datatype `Word8`, defined in `Data.Word`, represents unsigned 8-bit integers, with values ranging from 0 to 255. We can represent any nonnegative integer as a list of unsigned 8-bit integers. Any finite list $w_{n-1} \dots w_0$ represents the number $c = \sum_{0 \leqslant i < n} w_i \cdot 256^i$. (Note that $n$ is allowed to be 0 in the above, so the empty list of words represents the number 0.)

   Define the following function which computes the list of words corresponding to $n + 1$, where the input list corresponds to $n$:

```
import Data.Word ( Word8 )
incr :: [Word8] → [Word8]
```

   Here are some sample cases:

```
incr []                         = [1]
incr [0]                        = [1]
incr [22]                       = [23]
incr [254]                      = [255]
incr [255]                      = [1,0]
incr [255,22,23,255,255]        = [255,22,24,0,0]
incr [255,255,255,255,255]      = [1,0,0,0,0,0]
```

   (**Note:** Use **`toInteger`** to convert from `Word8` to **`Integer`** and **`fromIntegral`** to convert from **`Integer`** to `Word8`.)

4. The datatype `Int8`, defined in `Data.Int`, represents 8-bit signed integers, with values ranging from $-128$ to 127. These values are represented in a machine in **two's complement form**. Such a representation is a sequence $s = b_7 b_6 \dots b_0$ where each $b_i \in \{0, 1\}$, and its value is defined by:

$$val(s) = -b_7 \cdot 128 + \sum_{0 \leqslant i < 7} b_i \cdot 2^i$$

   Instead of representing integers in this range as a sequence of bits, we represent them as a list of length 8 of **`Bool`** values, with **`False`** and **`True`** representing 0 and 1 respectively. Define the following functions that convert from `Int8` to `[Bool]` and back. You can assume that the boolean lists provided as input are of length exactly 8.

```
import Data.Int ( Int8 )
twosVal :: [Bool] → Int8
twosRep :: Int8 → [Bool]
```

Here are some sample cases:

```
twosVal [False, False, False, False, False, False, False, False]    =    0
twosVal [False, False, False, False, False, False, False, True]     =    1
twosVal [False, True, True, False, True, False, False, False]       =  104
twosVal [False, True, True, True, True, True, True, True]           =  127
twosVal [True, False, False, False, False, False, False, False]     = -128
twosVal [True, False, False, True, True, False, False, False]       = -104
twosVal [True, True, True, False, True, False, False, False]        =  -24
twosVal [True, True, True, True, True, True, True, True]            =   -1

twosRep 0       = [False, False, False, False, False, False, False, False]
twosRep 1       = [False, False, False, False, False, False, False, True]
twosRep 104     = [False, True, True, False, True, False, False, False]
twosRep 127     = [False, True, True, True, True, True, True, True]
twosRep (-128)  = [True, False, False, False, False, False, False, False]
twosRep (-104)  = [True, False, False, True, True, False, False, False]
twosRep (-24)   = [True, True, True, False, True, False, False, False]
twosRep (-1)    = [True, True, True, True, True, True, True, True]
```

(**Note:** Use **`toInteger`** to convert from `Int8` to **`Integer`** and **`fromIntegral`** to convert from **`Integer`** to `Int8`.)

5. 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 other functions defined in `Data.Ratio`, like `numerator` and `denominator`, and the function `fromIntegral`. (Note: `numerator rat` can be positive or negative, but `denominator rat` is always positive.)

   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 $i \geqslant 1$. 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:

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

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 $\frac{3}{5}$. A continued fraction is represented in Haskell as a *nonempty* list of integers, all of whose elements are positive, except perhaps the first element.

Define the following functions that compute the rational number corresponding to a finite continued fraction, and produces a finite continued fraction representing a given rational. (Check with the given sample cases.)

```
computeRat :: [Integer] → Rational
cf :: Rational → [Integer]
computeRat [1,4,5]          = 26%21
computeRat [-1,1,1,1,1]     = (-2)%5
cf (26%21)                  = [1,4,5]
cf (-26%21)                 = [-2,1,3,5]
cf (42%31)                  = [1,2,1,4,2]
cf (-42%31)                 = [-2,1,1,1,4,2]
```

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

6. 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, $\sqrt{15}$ can be written as[1]

$$3 + \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{1 + \cfrac{1}{6 + \ldots}}}}$$

This is represented more succinctly as $[3; 1, 6, 1, 6, \ldots]$. If we truncate this list at some finite point, we get a finite rational approximation for $\sqrt{15}$.

Define the following function (with indicative sample cases provided) which computes a close rational approximation for $\sqrt{15}$.

```
approxTarget :: Double → Rational
approxTarget 0.0001          = 1677 % 433
approxTarget 0.0000000000001 = 50725923 % 13097377
```

The function should satisfy the following relation.

```
abs (sqrt 15 - computeFrac (approxTarget epsilon)) < epsilon    where
    computeFrac :: Rational → Double
    computeFrac x = fromIntegral (numerator x) / fromIntegral (denominator x)
```

**Note: Do not use** the `approxRational` function from `Data.Ratio`. Write a program that tries to compute more and more of the infinite continued fraction to get your approximation. Since there are multiple answers possible, the sample cases above are only indicative. We will check the correctness of your solution by actually calculating if the error is less than `epsilon`.

**Hint:** You will find it easier to find an approximation for $\sqrt{15} - 3$ and then adding 3 to the result.

[1] This can be verified by denoting the continued fraction as $x$ and observing that $x = 3 + \frac{1}{1 + \frac{1}{6 + (x-3)}}$, i.e. $x = 3 + \frac{1}{1 + \frac{1}{x+3}}$. Simplifying this, we get $x = \frac{4x + 15}{x + 4}$ and hence $x^2 + 4x = 4x + 15$, from which it follows that $x^2 = 15$, and thus $x = \sqrt{15}$.