National Undergraduate Programme in Mathematical Sciences

National Graduate Programme in Computer Science

Functional Programming in Haskell

Assignment 1

- **Due date**: September 10, 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!

---

1. Recall that $\binom{n}{r}$, pronounced "n choose r", is the number of r-element subsets of $\{1, \ldots, n\}$. Define a Haskell function

```
choose :: Integer → Integer → Integer
```

   such that `choose n r` computes $\binom{n}{r}$. We can assume that $0 \leqslant r \leqslant n$. Here are a few sample cases.

```
choose 10 0     = 1
choose 10 1     = 10
choose 10 3     = 120
choose 10 5     = 252
choose 10 7     = 120
choose 10 10    = 1
```

2. Define a Haskell function

```
primeSum :: Int → Integer
```

   such that `primeSum n` returns the sum of all prime numbers $\leqslant n$. Here are some sample cases.

```
primeSum (-100)     = 0
primeSum 0          = 0
primeSum 1          = 0
primeSum 2          = 2
```

```
primeSum 3          = 5
primeSum 7          = 17
primeSum 100        = 1060
primeSum 1000       = 76127
primeSum 10000      = 5736396
```

3. Define a Haskell function

```
leftRotate :: Integer → Integer
```

such that `leftRotate n` computes the *left rotation* of a given nonnegative integer n. The left rotation of such an n is achieved by removing the leftmost digit and placing it at the rightmost (and ignoring leading zeros). Sample cases:

```
leftRotate 2            = 2
leftRotate 200          = 2
leftRotate 203          = 32
leftRotate 5241093      = 2410935
```

Hint: The function `intRev` from the slides might help.

4. The *Collatz function* c is defined for positive integers as follows:

$$c(n) = \begin{cases} \dfrac{n}{2} & \text{if } n \text{ is even} \\ 3n+1 & \text{otherwise} \end{cases}$$

The *Collatz conjecture* asserts that for all positive n, there exists a nonnegative k such that $c^k(n) = 1$. (Here, $c^k(n)$ stands for $c(c(\cdots(c(c(n)))\cdots))$, where c is applied k times. By convention, $c^0(n) = n$.)

Define a Haskell function

```
cLength :: Integer → Maybe Int
```

where `cLength n` returns the least $k < 10000$ such that $c^k(n) = 1$. It should return **`Nothing`** if there is no such $k < 10000$. Here are a few sample cases.

```
cLength 1           = Just 0
cLength 2           = Just 1
cLength 128         = Just 7
cLength 256         = Just 8
cLength 85          = Just 9
cLength 100         = Just 25
cLength 1000        = Just 111
cLength 9999        = Just 91
```

```
cLength 10000       = Just 29
cLength (2^9999)    = Just 9999
cLength (2^10000)   = Nothing
```

5. A quintuple of integers (s,x,y,z,w) can be used to represent a real number with three decimal places, where s denotes the sign. For such a quintuple to be a valid representation of a real number, s should be either 1 or -1, x should be non-negative, and y, z and w should be single digits, and the number represented is $s \times x.yzw$. Write a program that finds the *fractional part* of a number $a = s \times x.yzw$. This is the unique nonnegative number $f < 1$ such that $a - f$ is an integer. When a has three decimal places, the fractional part can be represented as a triple of integers. Define a Haskell function

```
frac :: (Int, Int, Int, Int, Int) → Maybe (Int, Int, Int)
```

which returns the fractional part of a number represented by the input, if it is a valid representation. Otherwise it should return `Nothing`. Here are a few sample cases.

```
frac (1,0,-1,0,0)       = Nothing       -- y is negative
frac (-1,100,11,2,3)    = Nothing       -- y is not a single digit
frac (0,1,0,0,0)        = Nothing       -- s is neither 1 nor -1
frac (1,-2,1,1,1)       = Nothing       -- x is negative
frac (1,0,0,0,0)        = Just (0,0,0)
frac (1,100,0,0,0)      = Just (0,0,0)
frac (-1,100,0,0,0)     = Just (0,0,0)
frac (1,0,1,4,5)        = Just (1,4,5)
frac (1,100,1,4,5)      = Just (1,4,5)
frac (-1,100,1,4,5)     = Just (8,5,5)
frac (-1,2,2,0,0)       = Just (8,0,0)
frac (-1,0,2,0,0)       = Just (8,0,0)
```

6. For integers $n > 0$ and $b > 1$, the *integer logarithm* of n in base b is the largest integer k such that $b^k \leqslant n$. Define a Haskell function

```
ilog :: Integer → Integer → Integer
```

such that `ilog n b` computes the integer logarithm of `n` in base `b`. You can assume that `n > 0` and `b > 1`. Here are a few sample cases.

```
ilog 1 2            = 0
ilog 2 2            = 1
ilog 128 2          = 7
ilog 200 2          = 7
ilog 200 3          = 4
ilog 256 2          = 8
```

```
ilog 256 3          = 5
ilog 256 4          = 4
```

7. Recall the Taylor's series expansion for the cosine function.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Define two Haskell functions

```
cos1 :: Integer → Double → Double
cos2 :: Double → Double → Double
```

such that `cos1 n x` (for $n \geqslant 1$) computes the partial sum consisting of the first n terms in the Taylor's series for $\cos x$, and `cos2 e x` (for $e > 0$) computes some partial sum s of the Taylor's series for $\cos x$ such that $|s - \cos x| \leqslant e$. We present several sample cases below. Do not worry if you do not get the exact result as displayed below, as long as the difference is negligible. We will only check if your results match ours to some degree of precision.

```
cos1 1 0                = 1.0
cos1 10 0               = 1.0
cos1 100 (pi/4)         = 0.7071067811865475
cos1 1000 (pi/2)        = 4.264460367971268e-17
cos1 1000 pi            = -1.0000000000000002
cos2 1.0e-15 0          = 1.0
cos2 1.0e-15 pi         = -1.0000000000000004
cos2 1.0e-15 (pi/2)     = 6.092276318286374e-17
cos2 1.0e-15 (3*pi/4)   = -0.7071067811865475
cos2 1.0e-15 2          = -0.41614683654714246
```

Hint: Use the built-in Haskell function **`cos`** to verify the correctness of your function. One strategy for computing these functions is to repeatedly compute triples of the form (m, l, s), where l is the m^{th} term of the Taylor expansion for $\cos x$, and s is the sum of the first m terms. Think of what the values of l and s are when $m = 1$, and how you can compute the $m + 1^{\text{st}}$ triple easily from the m^{th} triple (you might need to use the **`fromInteger`** function for this). For `cos1 n x`, you should stop when you reach an entry of the form (n, l, s), and for `cos2 e x`, you should stop when you reach a triple (m, l, s) where $|s - \cos x| \leqslant e$.