Introduction to Programming: Assignment 1

Due: October 19, 2021. 11.59 am

Important Instructions:

- Submit your solution in a single file named `loginid.1.hs` on Moodle. For example, if I were to submit a solution, the file would be called `spsuresh.1.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.1.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. Define a function `nextSquare :: Integer -> Integer` such that `nextSquare n` returns the least square `m` with `m >= n`.

   Sample cases:

   ```
   nextSquare (-10)    = 0
   nextSquare 0        = 0
   nextSquare 1        = 1
   nextSquare 25       = 25
   nextSquare 36       = 36
   nextSquare 102      = 121
   ```

2. Define a function `previousCube :: Integer -> Integer` such that `previousCube n` returns the greatest cube `m` with `m <= n`, for `n > 0`. Assume that only positive inputs will be provided.

   Sample cases:

```
previousCube 1      = 1
previousCube 100    = 64
previousCube 1000   = 1000
previousCube 5000   = 4913
previousCube 10000  = 9261
```

3. Define a function `nextPalin :: Integer -> Integer` such that `nextPalin n` returns the least *palindromic number* `m` with `m >= n`.

   Sample cases:

```
nextPalin (-10)     = 0
nextPalin 0         = 0
nextPalin 5         = 5
nextPalin 55        = 55
nextPalin 56        = 66
nextPalin 102       = 111
nexPalin 5962       = 5995
```

4. Define a function `primesIn :: Integer -> Integer -> [Integer]` such that for any integers `l` and `u`, `primesIn l u` returns the list of all primes in the range `[l..u]`.

   Sample cases:

```
primesIn (-20)  (-10)   = []
primesIn (-20)  0       = []
primesIn (-10)  (-20)   = []
primesIn 100    50      = []
primesIn 50     50      = []
primesIn 53     53      = [53]
primesIn 50     100     = [53,59,61,67,71,73,79,83,89,97]
```

5. Define a function `isPrime :: Integer -> Bool` such that `isPrime n` returns `True` if `n` is prime, and `False` if not.

   Sample cases:

```
isPrime (-20)       = False
isPrime 0           = False
isPrime 1           = False
isPrime 2           = True
isPrime 5           = True
isPrime 27          = False
isPrime 47          = True
```

6. Define a function `decToBin :: Integer -> Integer` such that `decToBin n` returns the binary representation of `n`, for any `n >= 0`. Assume that only nonnegative inputs will be provided.

   Sample cases:

```
decToBin 0          = 0
decToBin 1          = 1
decToBin 25         = 11001
decToBin 63         = 111111
decToBin 64         = 1000000
decToBin 65         = 1000001
decToBin (2^24)     = 1000000000000000000000000
```

7. Define a function `binToDec :: Integer -> Integer` such that for any `n >= 0` which contains only the digits `0` and `1`, `binToDec n` outputs `m`, where `n` is the binary representation of `m`. Assume that all the inputs provided will be nonnegative and will only contain the digits `0` and `1`.

   Sample cases:

```
binToDec 0          = 0
binToDec 1          = 1
binToDec 11001      = 25
binToDec 111111     = 63
binToDec 1000000    = 64
binToDec 1000001    = 65
binToDec 1001100011 = 611
```