Introduction to Programming: Assignment 1

Due: August 31, 2022. 11.59 pm

Important Instructions: 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.

---

1. Define a function `isPOT :: Integer → Bool` that checks if the given input (a positive integer) is a power of 3.

   Sample cases:

   ```
   isPOT 1          = True
   isPOT 2          = False
   isPOT 3          = True
   isPOT 9          = True
   isPOT 21         = False
   isPOT 27         = True
   isPOT 81         = True
   ```

2. Define a function `isPPOT :: Integer → Bool` that checks if the given input (a positive integer) is a *prime* power of 3, i.e. the input is of the form 3^p for a prime p.

   Sample cases:

   ```
   isPPOT 1         = False
   isPPOT 2         = False
   isPPOT 3         = False
   isPPOT 9         = True
   isPPOT 21        = False
   isPPOT 27        = True
   isPPOT 81        = False
   ```

3. Define a function `intToOct :: Int → String` that produces the octal (base-8) representation of a non-negative integer.

   Sample cases:

   ```
   intToOct 2       = "2"
   intToOct 20      = "24"
   intToOct 200     = "310"
   ```

```
intToOct 2000         = "3720"
intToOct 20000        = "47040"
intToOct 200000       = "606500"
```

4. Define a function `octToInt :: String → Int` that produces the integer value of a given octal representation. We will assume that the string contains only digits from `0` to `7`. For instance `octToInt "606500"` should give the value `200000`.

5. Define a function `leftRotate :: Integer -> Integer` that computes the *left rotation* of a given nonnegative integer n. The left rotation of a non-negative integer 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
```

6. Define a function `rightRotate :: Integer -> Integer` that performs a right rotation, analogous to the previous problem.

   Sample cases:

```
rightRotate 2         = 2
rightRotate 200       = 20
rightRotate 203       = 320
rightRotate 2410935   = 5241093
```

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

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

   The *Collatz conjecture* asserts that for all positive n, there exists a nonnegative m such that $c^m(n) = 1$.

   Define a function `collatz :: Int → [Int]` which returns the *finite list* of all integers

$$\{c^m(n) \mid m \geqslant 0, \neg\exists k < m : (c^k(n) = 1)\},$$

   if n is positive.

   Sample cases:

```
collatz 1    = [1]
collatz 4    = [4,2,1]
collatz (-5) = []
collatz 0    = []
collatz 5    = [5,16,8,4,2,1]
collatz 22   = [22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
```

8. *The Josephus problem*: This problem has a number of warriors seated in a circle, numbered 1 to n clockwise. Warrior 1 kills 2, then 3 kills 4, then 5 kills 6 &c. till it comes all the way around. Then each warrior kills the nearest surviving person to his left, and this continues till only one warrior is left. The problem is to determine who survives at the end.

   For instance, if there are 10 warriors to start with, then the order in which the kills happen is given below (the pair (i, j) means that warrior i kills warrior j):

   $$(1,2),(3,4),(5,6),(7,8),(9,10),(1,3),(5,7),(9,1),(5,9).$$

   As we can see, 5 is the winner.

   As another example, if there are 13 warriors to start with, the order of kills is this:

   $$(1,2),(3,4),(5,6),(7,8),(9,10),(11,12),(13,1),(3,5),(7,9),(11,13),(3,7),(11,3).$$

   As we can see, 11 is the winner.

   Define a function `josephus :: Int → [(Int, Int)]` which on input n, gives the list of all kills that happen with n warriors (n is assumed to be positive).

   Define a function `josephusWinner :: Int → Int` that produces the lone survivor after all the kills have happened, starting with n warriors (n is again assumed to be positive).