Introduction to Programming: Assignment 1

Due: January 31, 2021. 11.59 pm

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`**. 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 **`isPrimePower :: Integer -> Bool`** that checks if the given input (a positive integer) is a power of a prime number.

   Sample cases:

   ```
   isPrimePower 1        = False
   isPrimePower 2        = True
   isPrimePower 5        = True
   isPrimePower 10       = False
   isPrimePower 81       = True
   isPrimePower 256      = True
   isPrimePower 1000     = False
   ```

2. 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"
   ```

3. 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`**.

4. 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

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

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]
```

5. *The Josephus problem*: This problem has a number of Twitter users seated in a circle, numbered 1 to n clockwise. They keep tweeting on politics, angering each other. Eventually they all get so angry that a war starts. User 1 gets user 2 suspended, then 3 gets 4 suspended, then 5 gets 6 suspended, etc. till it comes all the way around. Then each user gets the nearest surviving person to his left suspended, and this continues till only one user is left. The problem is to determine who survives at the end.

For instance, if there are 10 users to start with, then the order in which the suspensions happen is given below (the pair (i, j) means that user i gets user j suspended):

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

As we can see, 5 is the winner (but really the loser, since he has no one to read his tweets and outrage).

As another example, if there are 13 users to start with, the order of suspensions is as given below (with 11 ending up as the winner).

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

Define a function `josephus :: Int -> [(Int, Int)]` which on input *n*, gives the list of all suspensions that happen with *n* users (*n* is assumed to be positive).

Define a function `josephusWinner :: Int -> Int` that produces the lone survivor after all the suspensions have happened, starting with *n* users (*n* is again assumed to be positive).

6. Define the following type synonyms to represent the scores of a player in a game.

```
type Player       = String
type Points       = Int
type Assists      = Int
type Rebounds     = Int
type Stats        = (Points, Rebounds, Assists)
type GameScore    = (Player, Points, Rebounds, Assists)
```

We are given a list of scores by multiple players over several games. Here is an example list:

```
[("LBJ", 22, 5, 5), ("AD", 18, 7, 2), ("LBJ", 22, 7, 10),
 ("AD", 28, 8, 5), ("LBJ", 18, 9, 5), ("LBJ", 34, 6, 8),
 ("AD", 18, 9, 6)]
```

Write a function

```
compileStats :: [GameScore] -> [(Player, Stats)]
```

which computes the total points, rebounds, and assists scored by each player. For instance, the above list of game scores leads to the following stats.

```
[("AD", (64, 24, 13)), ("LBJ", (96, 27, 28))]
```

You might wonder about the order of the players in the output list. It does not matter what order your program produces them in. So for the above input, it is also acceptable to output the stats of LeBron James first, followed by those of Anthony Davis.