問題 6

次のように定義されたデータ型 Nat (自然数) において、商と余りを求める関数 divide と累乗を計算する関数 expt を定義しなさい。

Nat の定義

```
data Nat = Zero | Succ Nat
```

解答及び解説

関数 divide について

まずは、型 Nat の割り算を考える。足し算 plus を考えたときは、型 Nat のことばを使って、その規則を次のように表現した。

$$\begin{cases} \text{自然数0とmを足すと、結果はmである。} \\ \text{自然数nとmの足し算は、自然数「nの前の数」とmを足した結果の次の数に等しい。} \end{cases}$$

同様に、割り算の規則を (不完全ながら) 表現してみると、

自然数 n を m で割った商は、$n - m$ を m で割った商の次の数に等しい。

となる。このままプログラムにすれば、

```
div n m = Succ (div (minus n m) m)
```

となるが、自然数での引き算 minus は、部分関数であったことを思い出そう。(そのため、minus の計算結果を Maybe Nat 型として定義した。)

このことを考えると divide は、次のように定義することができる。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m = divide' (minus n m) m n


divide' :: Maybe Nat -> Nat -> Nat -> (Nat, Nat)
divide' Nothing m n = (Zero, n)
divide' (Just l) m n = let (q, r) = divide' (minus l m) m l in (Succ q, r)
```

ここで、divide' という関数を使って Maybe Nat 型を用いた計算をしているが、Maybe a 型の計算をするための関数 maybe が用意されているので、それを使うと次のようにできる。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m = maybe (Zero, n) div (minus n m)
    where div k = next (divide k m)
          next (q, r) = (Succ q, r)
```

関数 maybe の型は、

```
maybe :: b -> (a -> b) -> Maybe a -> b
```

であり、一つ目の引き数は Maybe a の値が Nothing の場合の値。二つ目の引き数は Maybe a の値が Just n の場合に n に適用する関数。今回の場合、型 a が Nat で、型 b は (Nat, Nat) となっている。また、第 2 引き数に使う関数 div は、上述の割り算の規則にある通り divide (minus n m) m の結果に、商の部分については次の数としたものが結果である。そこで、商と余りの組に対して、商を次の数にして余りはそのままの組を求める関数 next を用いている。

ところで、minus の結果が Nothing となるのは、「$n < m$」のときである。なので、型 Nat を Ord クラスのインスタンスにしておけば、大小の比較が可能になるので、次のようなプログラムでも良いだろう。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m | n < m     = (Zero, n)
           | otherwise = let (q, r) = divide (minus n m) m in (Succ q, r)
```

但し、この時に利用する minus は、結果が Nat 型がなるように定義する必要がある。

最後に、minus の定義を含めてまとめると、次のようになる。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m = maybe (Zero, n) div (minus n m)
    where div k = next (divide k m)
          next (q, r) = (Succ q, r)

minus :: Nat -> Nat -> Maybe Nat
minus n Zero = Just n
minus Zero m = Nothing
minus (Succ n) (Succ m) = minus n m
```

関数 expt について

次に、 自然数の累乗 (expt) を考える。累乗の規則は、次のようになる。

$$\begin{cases} \text{自然数nの0乗は、1である。} \\ \text{自然数nのm乗は、自然数nの「mの前の数」乗に、nを掛けた数に等しい。} \end{cases}$$

「m の前の数」を求める関数は無いため、プリント No.7 と同じように、パターンを使って「の前の数」を求めるようにする。

```
expt :: Nat -> Nat -> Nat
expt n Zero = Succ Zero
expt n (Succ m) = times n (expt n m)
```