



Ejercicio 1. Implementar la función `fibonacci :: Integer -> Integer` que devuelve el i -ésimo número de Fibonacci. Recordar que la secuencia de Fibonacci se define como:

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{en otro caso} \end{cases}$$

```
problema fibonacci (n: ℤ) : ℤ {
  requiere: { n ≥ 0 }
  asegura: { resultado = fib(n) }
}
```

Ejercicio 2. Implementar una función `parteEntera :: Float -> Integer` según la siguiente especificación:

```
problema parteEntera (x: ℝ) : ℤ {
  requiere: { True }
  asegura: { resultado ≤ x < resultado + 1 }
}
```

Ejercicio 3. Especificar e implementar la función `esDivisible :: Integer -> Integer -> Bool` que dados dos números naturales determinar si el primero es divisible por el segundo. No está permitido utilizar las funciones `mod` ni `div`.

Ejercicio 4. Especificar e implementar la función `sumaImpares :: Integer -> Integer` que dado $n \in \mathbb{N}$ sume los primeros n números impares. Por ejemplo: `sumaImpares 3` \rightsquigarrow `1+3+5` \rightsquigarrow `9`.

Ejercicio 5. Implementar la función `medioFact :: Integer -> Integer` que dado $n \in \mathbb{N}$ calcula $n!! = n(n-2)(n-4)\dots$.

```
problema medioFac (n: ℤ) : ℤ {
  requiere: { n ≥ 0 }
  asegura: { resultado = ∏i=0⌊ $\frac{n-1}{2}$ ⌋ (n - 2i) }
}
```

Por ejemplo:

`medioFact 10` \rightsquigarrow `10 * 8 * 6 * 4 * 2` \rightsquigarrow `3840`.

`medioFact 9` \rightsquigarrow `9 * 7 * 5 * 3 * 1` \rightsquigarrow `945`.

`medioFact 0` \rightsquigarrow `1`.

Ejercicio 6. Especificar e implementar la función `sumaDigitos :: Integer -> Integer` que calcula la suma de dígitos de un número natural. Para esta función pueden utilizar `div` y `mod`.

Ejercicio 7. Implementar la función `todosDigitosIguales :: Integer -> Bool` que determina si todos los dígitos de un número natural son iguales, es decir:

```
problema todosDigitosIguales (n: ℤ) : ℬ {
  requiere: { n > 0 }
  asegura: { resultado = true ↔ todos los dígitos de n son iguales }
}
```

Ejercicio 8. Implementar la función `iesimoDigito :: Integer -> Integer -> Integer` que dado un $n \in \mathbb{N}_{\geq 0}$ y un $i \in \mathbb{N}$ menor o igual a la cantidad de dígitos de n , devuelve el i -ésimo dígito de n .

```
problema iesimoDigito (n: ℤ, i: ℕ) : ℤ {
  requiere: { n ≥ 0 ∧ 1 ≤ i ≤ cantDigitos(n) }
  asegura: { resultado = (n div 10cantDigitos(n)-i) mod 10 }
}

problema cantDigitos (n: ℤ) : ℕ {
  requiere: { n ≥ 0 }
  asegura: { n = 0 → resultado = 1 }
  asegura: { n ≠ 0 → (n div 10resultado-1 > 0 ∧ n div 10resultado = 0) }
}
```

Ejercicio 9. Especificar e implementar una función `esCapicua :: Integer -> Bool` que dado $n \in \mathbb{N}_{\geq 0}$ determina si n es un número capicúa.

Ejercicio 10. Especificar, implementar y dar el tipo de las siguientes funciones (simil Ejercicio 4 Práctica 2 de Álgebra 1).

$$\begin{array}{ll} \text{a) } f1(n) = \sum_{i=0}^n 2^i, n \in \mathbb{N}_0. & \text{c) } f3(n, q) = \sum_{i=1}^{2n} q^i, n \in \mathbb{N}_0 \text{ y } q \in \mathbb{R} \\ \text{b) } f2(n, q) = \sum_{i=1}^n q^i, n \in \mathbb{N} \text{ y } q \in \mathbb{R} & \text{d) } f4(n, q) = \sum_{i=n}^{2n} q^i, n \in \mathbb{N}_0 \text{ y } q \in \mathbb{R} \end{array}$$

Ejercicio 11. a) Especificar e implementar una función `eAprox :: Integer -> Float` que aproxime el valor del número e a partir de la siguiente sumatoria:

$$\hat{e}(n) = \sum_{i=0}^n \frac{1}{i!}$$

b) Definir la constante `e :: Float` como la aproximación de e a partir de los primeros 10 términos de la serie anterior.
¡Atencion! A veces ciertas funciones esperan un `Float` y nosotros tenemos un `Int`. Para estos casos podemos utilizar la función `fromIntegral :: Int -> Float` definida en el Preludio de Haskell.

Ejercicio 12. Para $n \in \mathbb{N}$ se define la sucesión:

$$a_n = 2 + \frac{1}{2 + \frac{1}{\ddots \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}}} \quad (\text{aparece } n \text{ veces el } 2).$$

Lo cual resulta en la siguiente definición recursiva: $a_1 = 2, a_n = 2 + \frac{1}{a_{n-1}}$. Utilizando esta sucesión, especificar e implementar una función `raizDe2Aprox :: Integer -> Float` que dado $n \in \mathbb{N}$ devuelva la aproximación de $\sqrt{2}$ definida por $\sqrt{2} \approx a_n - 1$. Por ejemplo:

```
raizDe2Aprox 1 ~ 1
raizDe2Aprox 2 ~ 1,5
raizDe2Aprox 3 ~ 1,4
```

Ejercicio 13. Especificar e implementar la siguiente función:

$$f(n, m) = \sum_{i=1}^n \sum_{j=1}^m i^j$$

Ejercicio 14. Especificar e implementar una función `sumaPotencias :: Integer -> Integer -> Integer -> Integer` que dados tres naturales q, n, m sume todas las potencias de la forma q^{a+b} con $1 \leq a \leq n$ y $1 \leq b \leq m$.

Ejercicio 15. Especificar e implementar una función `sumaRacionales :: Integer -> Integer -> Float` que dados dos naturales n, m sume todos los números racionales de la forma p/q con $1 \leq p \leq n$ y $1 \leq q \leq m$, es decir:

```
problema sumaRacionales (n : ℕ, m : ℕ) : ℝ {
  requiere: { True }
  asegura: { resultado =  $\sum_{p=1}^n \sum_{q=1}^m \frac{p}{q}$  }
}
```

Ejercicio 16. Recordemos que un entero $p > 1$ es **primo** si y sólo si no existe un entero k tal que $1 < k < p$ y k divida a p .

- Implementar `menorDivisor :: Integer -> Integer` que calcule el menor divisor (mayor que 1) de un natural n pasado como parámetro.
- Implementar la función `esPrimo :: Integer -> Bool` que indica si un número natural pasado como parámetro es primo.
- Implementar la función `sonCoprimos :: Integer -> Integer -> Bool` que dados dos números naturales indica si no tienen algún divisor en común mayor estricto que 1.
- Implementar la función `nEsimoPrimo :: Integer -> Integer` que devuelve el n -ésimo primo ($n \geq 1$). Recordar que el primer primo es el 2, el segundo es el 3, el tercero es el 5, etc.

Ejercicio 17. Implementar la función `esFibonacci :: Integer -> Bool` según la siguiente especificación:

```
problema esFibonacci (n: ℤ) : ℬ {
  requiere: { n ≥ 0 }
  asegura: { resultado = true ↔ n es algún valor de la secuencia de Fibonacci definida en el ejercicio 1 }
}
```

Ejercicio 18. Implementar una función `mayorDigitoPar :: Integer -> Integer` según la siguiente especificación:

```
problema mayorDigitoPar (n: ℕ) : ℕ {
  requiere: { True }
  asegura: { resultado es el mayor de los dígitos pares de n. Si n no tiene ningún dígito par, entonces resultado es -1. }
}
```

Ejercicio 19. Implementar la función `esSumaInicialDePrimos :: Int -> Bool` según la siguiente especificación:

```
problema esSumaInicialDePrimos (n: ℤ) : ℬ {
  requiere: { n ≥ 0 }
  asegura: { resultado = true ↔ n es igual a la suma de los m primeros números primos, para algún m. }
}
```

Ejercicio 20. Especificar e implementar la función `tomaValorMax :: Int -> Int -> Int` que dado un número entero $n_1 \geq 1$ y un $n_2 \geq n_1$ devuelve algún m entre n_1 y n_2 tal que $\text{sumaDivisores}(m) = \max\{\text{sumaDivisores}(i) \mid n_1 \leq i \leq n_2\}$

Ejercicio 21. Especificar e implementar una función `pitagoras :: Integer -> Integer -> Integer -> Integer` que dados $m, n, r \in \mathbb{N}_{\geq 0}$, cuente cuántos pares (p, q) con $0 \leq p \leq m$ y $0 \leq q \leq n$ satisfacen que $p^2 + q^2 \leq r^2$. Por ejemplo:

```
pitagoras 3 4 5 ≈ 20
```

```
pitagoras 3 4 2 ≈ 6
```