Задание 2. Итерационные алгоритмы и рекурсия

Задание 1

Как известно, $arctg\ x$ (арктангенс) – тригонометрическая функция, обратная к тангенсу.

$arctg\ x$ – бесконечно дифференцируемая функция, а значит, её можно разложить в бесконечную сумму степенных функций (см. ряд Тейлора):

$$arctg\ x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \cdots.$$

Требуется написать функцию `atan'`, вычисляющую арктангенс в точке x по формуле разложения в ряд Тейлора с заданной точностью $\varepsilon > 0$.

Задание 2

Математики доказали:

если f – функция, непрерывная на отрезке $[a, b]$, и на концах отрезка принимает значения разного знака: $sign(f(a)) \neq sign(f(b))$, то где-то внутри отрезка $[a, b]$ имеется точка c, такая, что $f(c) = 0$.

Это следствие из теоремы о промежуточном значении, также известной как теорема Больцано-Коши, позволяет приближенно (с заданной точностью $\varepsilon > 0$) находить корни уравнения $f(x) = 0$ для любой функции $f(x)$, лишь бы она была непрерывной, и были известны две точки a и b, в которых $f(a)$ и $f(b)$ имеют разный знак.

Сформулируем алгоритм поиска корней уравнения $f(x) = 0$ на отрезке $[a, b]$ с заданной точностью $\varepsilon > 0$:

1. Определим середину отрезка: $c = \frac{a+b}{2}$.
2. Если $|f(c)| < \varepsilon$ (предполагается что ε – некоторое очень малое дробное число), то решение уравнения найдено – это точка c.
3. Иначе рассматриваются два отрезка: $[a, c]$ и $[b, c]$. Из них выбирается такой, что функция в граничных точках имеет разный знак, *корень*

ищется внутри нового отрезка.

Предлагается:

1. написать функцию `solver`:
   - имеющую сигнатуру `solver :: (Double -> Double) -> Double -> Double -> Double -> Double`;
   - принимающую первым параметром функцию $f(x)$, вторым и третьим параметром границы отрезка $[a, b]$, четвертым параметром – число ε;
   - осуществляющую поиск корня уравнения $f(x) = 0$ с помощью представленного выше алгоритма.
2. использовать `solver` для нахождения корней уравнения $2^x = x^2$ при $x > 0$ и $x < 0$.

FYI: в базовой библиотеке языка Haskell есть полезная функция `signum`...

Задание 3

Последовательность чисел Фибоначчи задается рекуррентным соотношением:

$$F_0 = 1,\ F_1 = 1,$$
$$F_n = F_{n-1} + F_{n-2},\ n \geq 2.$$

Реализация функции получения n-го числа Фибоначчи, основанная на прямом рекурсивном определении, записывается элементарно:

```
fib 0 = 1
fib 1 = 1
fib n | n < 0 = error "n must be greater than or equal to 0"
fib n = fib (n - 1) + fib (n - 2)
```

Однако, функция `fib` является крайне неэффективной – количество её вызовов растёт экспоненциально с ростом значения аргумента.

Предлагается написать более эффективную реализацию, имеющую линейную сложность (по числу рекурсивных вызовов), используя механизм аккумуляторов и хвостовую рекурсию.

FYI: в GHCi встроен инструмент, позволяющий оценивать использование памяти и затраты времени на вычисление выражения:

```
GHCI> :set +s
GHCI> fib 30
```

```
1346269
(6.79 secs, 929,807,368 bytes)
```