Учреждение образования "Полоцкий Государственный Университет"

Факультет информационных технологий
Кафедра вычислительных систем и сетей

Отчет по Контрольной работе

по дисциплине: "Функциональное программирование"

ВЫПОЛНИЛ студент группы 18-ИТ-1
Дёкин А.Ю.
вариант № 1

ПРОВЕРИЛ Деканова М.В.

Полоцк 2020 г.

1 Задания

Задание 1.

Функция amember :: Double -> Double -> Int -> Double должна возвращать n-ый член арифметической прогрессии (члены прогрессии нумеруются с нуля). Аргументами функции являются первый член прогрессии, шаг, n. Например, в результате вызова amember 1, 2, 5 должно получиться число 11.

Задание 2.

Функция root :: Double -> Double должна вычислить приближенное значение корня уравнения cos x = x с точностью, заданной первым (и единственным) аргументом функции.

Задание 3.

Функция maxthree :: [Integer] -> [Integer] получает список целых и выдает список той же длины, содержащий в качестве i-го элемента максимальные значения трех элементов списка - i-го, (i-1)-го и (i+1)-го. Например, maxthree [3, 8, 6, 5, 1] => [8, 8, 8, 6, 5].

Задание 4.

Г). Функция split :: String -gt;Char -gt; [String] разрезает исходную строку на кус-ки, разделенные в исходной строке символом, указанным в качестве второго ар-гумента функции. Например, Например, (split "one,two„three,“,’) => ["one "two , "three]

Задание 5.

Функция fluffy :: Tree a -> Int вычисляет максимальный показатель ¾ветвисто-сти¿ узлов дерева. Показатель ¾ветвистости¿ узла - это количество поддеревьев этого узла.

Задание 6.

Составить бесконечный список частичных сумм ряда, представляющего собой разложение числа e, полученное подстановкой единицы в ряд Тейлора для экспоненты. Написать функцию для получения n-го члена этой последовательности.

2 Скриншоты

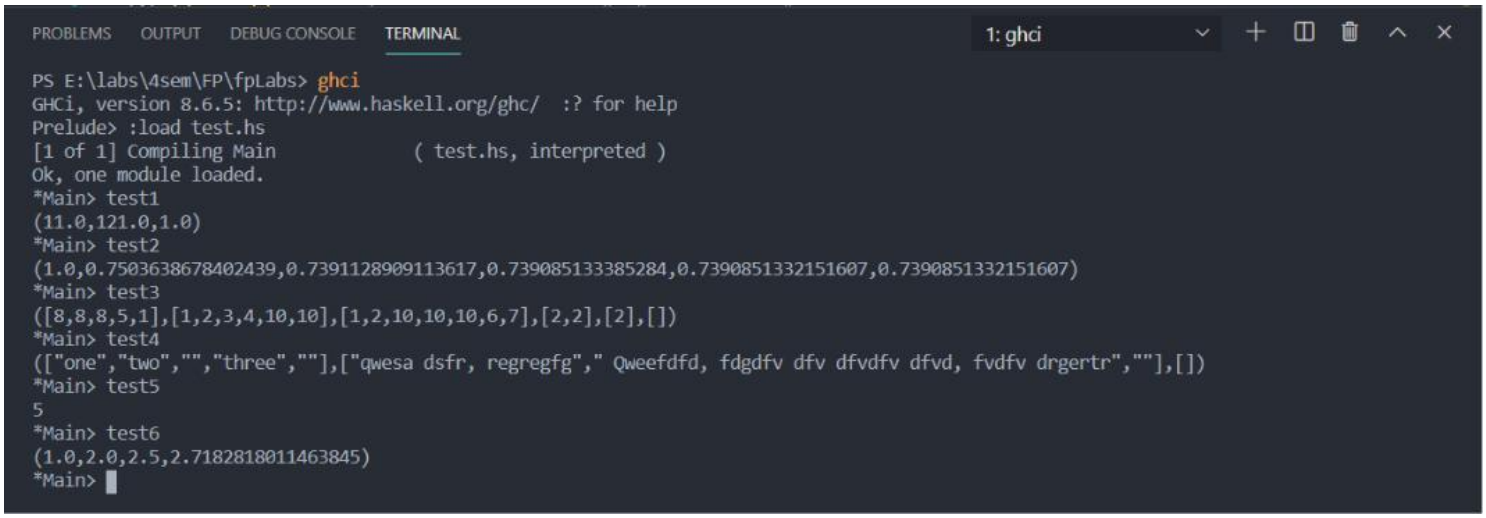

Рис. 1: Тестирование написанных функций

3 Source Code

test.hs

```haskell
--импорт необходимых библиотек
import Data.Maybe
import Data.List

--1е задание. Нахождение n-ого члена последовательности
amember :: Double -> Double -> Int -> Double --принимает 3 аргумента(a0, шаг, n-ый член)
amember a1 dt n = a1 + dt * fromIntegral(n)--формула для вычисления n-
ого члена последовательности

test1 = (amember 1 2 5, amember 1 10 12, amember 1 2 0) -- тест для первого задания

--вспомогательная ф-ия для 2-го задания
--принимает предыдущее приближенное значение и степень приближенности вычисления
f :: Double -> Double -> Double
f x 0 = x -- при степени приближенности равной 0 возвращается посленее вычисленное значение
f x n = f(x - ((x-cos(x))/(1+sin(x)))) (n - 1) --
 иначе вычисляем значение по формуле, уменьшая степень приближения

--основная ф-ия 2го задания, принимающая степень приближенннности вычисления
root :: Double -> Double
root 0 = 1 -- 0-й член последовательности = 1
root n = f 1 n -- иначе возвращаем значения, вычисленное вспомогательной ф-ией, передав 1-
й член = 1

test2 = (root 0, root 1, root 2, root 3, root 4, root 5) --тест для 2го задания

--3ее задание. Замена соседних элементов максимального эл-та списка максимальным значением
-- Алгоритм: елси синдекс след. эл-та равен индексу максимального -
 1 или +1, то мы заменяем его.
-- используется ф-ия map, которая делает обход по списку и выполняет указанный выше алгоритм
maxThree :: [Int] -> [Int]--принимает список целых и возвращает список целых
maxThree [] = [] --при передаче пустого списка возвращает пустой список
maxThree (x:xs) = map (\ t -> if(fromJust(elemIndex t (x:xs)) == (maxElIndex - 1)
    || fromJust(elemIndex t (x:xs)) == (maxElIndex + 1)) then maximum (x:xs) else t) (x:xs)
    where maxEl = maximum(x:xs) --вычисление максимального эл-та списка
          maxElIndex = fromJust(elemIndex (maxEl) (x:xs)) -- получение индекса максимального эл-
та

--тест для 3го задания
test3 = (maxThree [3, 8, 6, 5, 1],
         maxThree[1,2,3,4,5,10],
         maxThree[1,2,3,10,5,6,7],
         maxThree [1, 2],
         maxThree [2],
         maxThree [])

--всмогательная ф-ия для 4-го задания
```

```haskell
--возвращает строку, стоящую перед разделяющим символом
getStringBeforeRegex :: String -> Char -> String
getStringBeforeRegex [] _ = []
getStringBeforeRegex (x:xs) regex = if(x /= regex) then x : getStringBeforeRegex xs regex else g
etStringBeforeRegex [] regex

--всмогательная ф-ия для 4-го задания
--разбивает строку на нужные подстроки, но без первой подстроки
splitWithoutFisrt :: String -> Char -> [String]
splitWithoutFisrt [] _ = []
splitWithoutFisrt (x:xs) regex = if(x == regex) --если был встречен разделяющий символ
     then getStringBeforeRegex (xs) regex : splitWithoutFisrt xs regex --
возвращает подстроку до следующего разделяющего символа и продолжает выполнение
     else splitWithoutFisrt xs regex --продолжает искать разделяющий символ

--ф-ия разбивающая строк, используя переданный ей символ
split :: String -> Char -> [String]
split [] _ = [] -- передана пустая строка - возвращает пустой список
split str regex = getStringBeforeRegex str regex : splitWithoutFisrt str regex --добавляет 1-
ую подстроку к списку разбитых строк

--тест для 4го задания
test4 = (split "one,two,,three," ',', split "qwesa dsfr, regregfg. Qweefdfd, fdgdfv dfv dfvdfv d
fvd, fvdfv drgertr." '.', split "" '_')

--5-е задание
-- Описание стркутуры данных дерево
data BinaryTree a =
    Empty--есть пустые узлы
    | Node (BinaryTree a) a (BinaryTree a) -- есть поддеревья
    | Leaf a --конечные, непустые  эл-ты дерева
    deriving (Show) --наследование св-ва класса для отображения

{-Ф-ия для 5го задания
--Если есть поддеревья, считает максимальную ветвистость каждого(левого и правого)
--и выбирает максимальное, прибавив  к результату 1, так как учитывается корень -}
fluffy :: BinaryTree a -> Int --получает дерево и возвращает максималькое кол-во поддеревьев
fluffy Empty = 0 --если дерево пустое - 0
fluffy (Leaf a) = 1 --если имеется один эл-т - 1
fluffy (Node leftSubTree a rightSubTree) = 1 + max (fluffy leftSubTree) (fluffy rightSubTree)
```

```haskell
test5 = fluffy (Node(Node (Node (Node (Leaf (10)) 7 Empty) 6 Empty) 3 (Leaf(5))) 1 (Leaf (2)))
{-

               1
              / \
             3   2
           /  \
          6    5
        /  \
       7
      /
     10

bt = Node(Node (Node (Node (Leaf (10)) 7 Empty) 6 Empty) 3 (Leaf(5))) 1 (Leaf (2))

 -}


--6-е задание. Получение n-ого элемента ряда Тейлора для e
getNthElemFromTaylorSeries :: Int -> Double
factorial 0 = 1 --вспомогательная ф-ия факториала
factorial x = x * factorial(x-1)
getNthElemFromTaylorSeries 0 = 1 --0-й элемент = 1(по формуле)
getNthElemFromTaylorSeries n = 1/factorial(fromIntegral(n)) + getNthElemFromTaylorSeries (n-1)--
вычислени n-ого члена ряда и вычисление предыдущих

--Тест для 6го задания
test6 = (getNthElemFromTaylorSeries 0,
        getNthElemFromTaylorSeries 1,
        getNthElemFromTaylorSeries 2,
        getNthElemFromTaylorSeries 10)
```