ФГАОУ ВО «Севастопольский государственный университет»

Институт информационных технологий   
и управления в технических системах

ОТЧЁТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Исследование способов реализации рекурсивных алгоритмов   
в функциональном программировании»

по дисциплине «Парадигмы современных языков программирования»

Выполнила:  
студентка группы ИС/м-21-1-з  
Ускова Екатерина Дмитриевна

Севастополь

2022

# ЦЕЛЬ РАБОТЫ

Исследовать различные способы реализации рекурсивных алгоритмов в функциональном программировании. Реализовать и исследовать рекурсивные функции обработки списков на языке Scala.

# Ход работы

Написана функция, которая возвращает новый список, в котором каждый элемент является суммой предыдущего элемента нового списка и текущего элемента входного списка.

def cumsumList(lst: List[Int], sum: Int = 0): List[Int] = {

if (lst == Nil) Nil

else sum + lst.head :: cumsumList(lst.tail, sum + lst.head)

}

Также написана функция, выполняющая то же, но с использованием хвостовой рекурсии.

def cumsumListT(lst: List[Int]): List[Int] = {  
 def cumsumListAcc(\_lst: List[Int], sum: Int, acc: List[Int]): List[Int] = {  
 if (\_lst == Nil) acc  
 else cumsumListAcc(\_lst.tail, sum + \_lst.head, sum + \_lst.head :: acc)  
 }  
  
 cumsumListAcc(lst, 0, Nil).reverse

}

При достаточно большом количестве элементов в списке (более 6500) обычная рекурсия выдаёт ошибку переполнения стека (stack overflow), а хвостовая рекурсия продолжает работать.

# Вывод

Выполняя эту работу, я исследовала различные способы реализации рекурсивных алгоритмов в функциональном программировании. Реализовала и исследовала рекурсивные функции обработки списков на языке Scala.

# Приложение А

Листинг программного кода

package main.scala  
  
object Application {  
 // Функция возвращает новый список, в котором каждый элемент является  
 // суммой предыдущего элемента нового списка и текущего элемента входного списка  
 def cumsumList(lst: List[Int], sum: Int = 0): List[Int] = {  
 if (lst == Nil) Nil  
 else sum + lst.head :: cumsumList(lst.tail, sum + lst.head)  
 }  
  
 def cumsumListT(lst: List[Int]): List[Int] = {  
 def cumsumListAcc(\_lst: List[Int], sum: Int, acc: List[Int]): List[Int] = {  
 if (\_lst == Nil) acc  
 else cumsumListAcc(\_lst.tail, sum + \_lst.head, sum + \_lst.head :: acc)  
 }  
  
 cumsumListAcc(lst, 0, Nil).reverse  
 }  
  
 def main (args: Array[String]): Unit = {  
 scala.util.Random.setSeed(42)  
 var cnt = 0  
 var llst :List[Int] = List()  
 while(cnt<10000) {  
 llst = scala.util.Random.nextInt(10) :: llst  
 cnt += 1  
 }  
  
 //println(cumsumList(llst))  
 println(cumsumListT(llst))  
 }  
}