

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«Санкт-Петербургский политехнический университет Петра Великого»

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

**Курсовой проект  
по дисциплине «Функциональное  
программирование»**

Выполнила студентка гр. 3530904/80001:

Прохорова А. И.

Руководитель  
ассистент ВШПИ

Лукашин А. А.

Санкт-Петербург  
2019

# 1 Задание

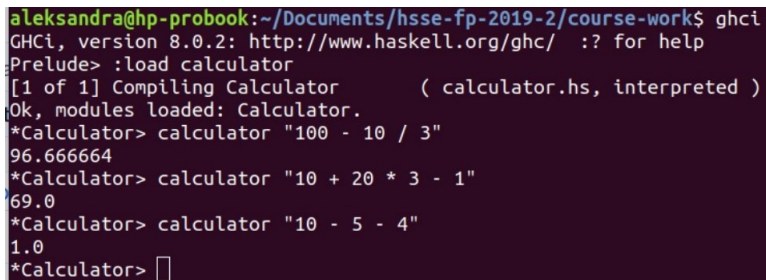
Калькулятор, поддерживающий простые арифметические операции и приоритеты.

## 2 Ход работы

### 2.1 Алгоритм решения

Расчёт выражения производится с помощью алгоритма сортировочной станции - способа разбора математических выражений, записанных в инфиксной нотации. Код программы приведён в приложении.

### 2.2 Скриншот



```
aleksandra@hp-probook:~/Documents/hsse-fp-2019-2/course-work$ ghci
GHCi, version 8.0.2: http://www.haskell.org/ghc/  :? for help
Prelude> :load calculator
[1 of 1] Compiling Calculator      ( calculator.hs, interpreted )
Ok, modules loaded: Calculator.
*Calculator> calculator "100 - 10 / 3"
96.6666664
*Calculator> calculator "10 + 20 * 3 - 1"
69.0
*Calculator> calculator "10 - 5 - 4"
1.0
*Calculator> 
```

## 3 Выводы

В ходе работы был изучен функциональный подход к программированию, который значительно отличается от стандартного императивного подхода. Изучены некоторые основные алгоритмы, используемые в функциональном программировании и произведена работа с ними.

## 4 Приложение

### 4.1 Код модуля calculator.hs

```
1 module Calculator where
2 import Data.List
3 import Data.Char
4
5 calculator :: String -> Float
6 calculator expr = processToken [] [] (words expr)
7
8 processToken :: [Float] -> [String] -> [String] -> Float
9
10 processToken numbers [] []
11   | (length numbers == 1) = head numbers
12   | otherwise = error "Error while counting expression"
13
14 processToken numbers operations [] =
15   processToken (count n2 n1 op : restN) restOp []
16   where op = head operations
17         restOp = tail operations
18         n1 = head numbers
19         n2 = head (tail numbers)
20         restN = tail (tail numbers)
21
22 processToken numbers operations tokens
23   | (isOp t == True) =
24     if (lenOp == 0) then processToken numbers (t : operations) (tail tokens)
25     else if (prior op >= prior t) then processToken (count n2 n1 op : restN) (t :
26       restOp) (tail tokens)
27     else processToken numbers (t : operations) (tail tokens)
28   | otherwise = processToken ((read t :: Float) : numbers) operations (tail tokens)
29   where t = head tokens
30         op = head operations
31         restOp = tail operations
32         n1 = head numbers
33         n2 = head (tail numbers)
34         restN = tail (tail numbers)
35         lenOp = length operations
36
37 count :: Float -> Float -> String -> Float
38 count a b "+" = a + b
39 count a b "-" = a - b
40 count a b "*" = a * b
41 count a b "/" = a / b
42 count a b operation = error "Operation not permitted"
43
44 isOp :: String -> Bool
45 isOp "+" = True
46 isOp "-" = True
47 isOp "*" = True
48 isOp "/" = True
49 isOp op = False
50
51 prior "*" = 2
52 prior "/" = 2
53 prior "+" = 1
54 prior "-" = 1
55 prior op = 0
```