

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

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

Курсовой проект
по дисциплине «Функциональное
программирование»

Выполнил студент гр. 3530904/80001:

Барсков А. М.

Руководитель
ассистент ВШПИ

Лукашин А. А.

Санкт-Петербург
2019

1 Задание

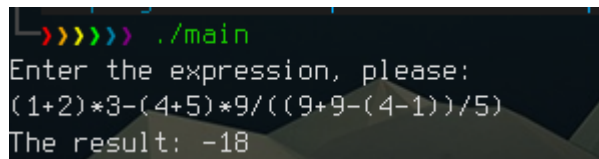
Калькулятор, поддерживающий простые арифметические операции, приоритеты и скобки.

2 Ход работы

2.1 Алгоритм решения

Выражение, заданное в инфиксной нотации с помощью алгоритма сортировочной станции (**Shunting Yard Algorithm**) переводится в обратную польскую нотацию, а затем вычисляется с помощью левоассоциативной свёртки получившегося списка токенов. Код программы приведён в приложении.

2.2 Скриншот

A screenshot of a terminal window with a dark background. The prompt is a multi-colored shell symbol followed by './main'. The user has entered the expression '(1+2)*3-(4+5)*9/((9+9-(4-1))/5)'. The program has responded with 'The result: -18'.

```
./main
Enter the expression, please:
(1+2)*3-(4+5)*9/((9+9-(4-1))/5)
The result: -18
```

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 файла main.hs

```
1 module Main where
2
3 import Calculator
4
5 main :: IO()
6 main = do
7     putStrLn "Enter the expression, please: "
8     input <- getLine
9     putStr "The result: "
10    print $ calculate input
```

4.2 Код модуля calc.hs

```
1 module Calculator
2 ( calculate
3 , calculateRPN
4 , convertToRPN
5 ) where
6
7 import Data.Char
8 import Data.List
9
10 calculate :: String -> Int
11 calculate [] = 0
12 calculate expr = calculateRPN $ convertToRPN expr
13
14 calculateRPN :: String -> Int
15 calculateRPN expr = head (foldl compute [] expr)
16     where
17         compute (x:y:zs) '+' = (x + y):zs
18         compute (x:y:zs) '-' = (y - x):zs
19         compute (x:y:zs) '*' = (x * y):zs
20         compute (x:y:zs) '/' = (y `div` x):zs
21         compute zs digit    = (digitToInt digit):zs
22
23 convertToRPN :: String -> String
24 convertToRPN tokens = shuntingYard tokens [] []
25
26 shuntingYard :: String -> String -> String -> String
27 shuntingYard [] [] out = reverse out
28 shuntingYard [] (op:ops') out = shuntingYard [] ops' (op:out)
29 shuntingYard (token:tokens) ops out
30     | isDigit token = shuntingYard tokens ops (token:out)
31     | isOperator token = case ops of
32         [] -> shuntingYard tokens (token:ops) out
33         (op':ops') -> if ((opPrecedence token) <= (opPrecedence op'))
34             then shuntingYard (token:tokens) ops' (op':out)
35             else shuntingYard tokens (token:ops) out
36     | isLeftParen token = shuntingYard tokens (token:ops) out
37     | isRightParen token = case ops of
38         ('(:ops') -> shuntingYard tokens ops' out
39         (op:ops') -> shuntingYard (token:tokens) ops' (op:out)
40
41 isOperator :: Char -> Bool
42 isOperator '+' = True
43 isOperator '-' = True
44 isOperator '*' = True
```

```
45 isOperator '/' = True
46 isOperator _   = False
47
48 isLeftParen :: Char -> Bool
49 isLeftParen '(' = True
50 isLeftParen _   = False
51
52 isRightParen :: Char -> Bool
53 isRightParen ')' = True
54 isRightParen _   = False
55
56 opPrecedence :: Char -> Int
57 opPrecedence '+' = 2
58 opPrecedence '-' = 2
59 opPrecedence '*' = 3
60 opPrecedence '/' = 3
61 opPrecedence _   = 0
```