Отчёты к лабораторным работам

по дисциплине  
«Тестирование программного обеспечения»

выполнил   
студент гр. ИС/б-18-1-з Демиденко А. А.  
зачётная книжка № 481483  
принял Строганов В. А.

Лабораторной работа № 1  
«Исследование способов анализа областей эквивалентности и построения тестовых последовательностей»

1. **Цель работы**

Исследовать способы анализа областей эквивалентности входных данных для тестирования программного обеспечения. Приобрести практические навыки составления построения тестовых последовательностей.

1. **Постановка задачи**

По варианту задаются требования к программам. Для каждой из них необходимо:

1. Написать программу, выполняющую заданные действия.
2. Определить области эквивалентности входных данных.
3. Составить примеры тестовых последовательностей.

Вариант: 83 `mod` 30 + 1 = 24

Задача 1. Дана квадратная матрица. Выполнить поворот этой матрицы на 180×k градусов, где k − целое число.

Задача 2. Дана строка. Выяснить, верно ли, что в строке имеются три идущих подряд буквы 'а'.

Задача 3. Программа, которая подсчитывает количество непустых строк в текстовом файле.

1. **Программный код**

Задача 1.

type Row a = [a]

type Matrix a = [Row a]

rotate :: Int -> Matrix a -> Matrix a

rotate i m

| i > 0 = rotate (i - 1) (reverse $ map reverse m)

| otherwise = m

Задача 2.

check3a :: String -> Bool

check3a (x1:x2:x3:\_) = all (== 'a') [x1, x2, x3]

check3a \_ = False

Задача 3.

countLines :: FilePath -> IO Int

countLines fileName = do

content <- lines <$> readFile fileName

return . length $ filter (not . null) content

1. **Описание областей эквивалентности**

|  |  |
| --- | --- |
| Значение k | Результат |
| нечётное | Выходное значение отлично от входного |
| чётное | Выходное значение равно входному |

Задача 1.

|  |  |
| --- | --- |
| Длина последовательности | Результат |
| ноль элементов | False |
| один элемент | False |
| два элемента | False |
| три элемента | True |
| три элемента | False |
| больше трёх элементов | True |
| больше трёх элементов | False |

Задача 2.

|  |  |
| --- | --- |
| Количество строк | Результат |
| ноль строк | 0 |
| одна строка | 1 |
| одна строка | 0 |
| больше одной строки | 1 и более |
| больше одной строки | 0 |

Задача 3.

1. **Примеры тестовых последовательностей**

Задача 1.

input1 :: Matrix Int

input1 =

[ [ 1, 2, 3, 4 ]

, [ 5, 6, 7, 8 ]

, [ 9, 0, 1, 2 ]

, [ 3, 4, 5, 6 ]

]

Задача 2.

“aaa”, “aaab”, “aba”, “”

Задача 3.

“\n”, “1\n2\n\n4”, “1”

**Выводы**

В ходе лабораторной работы было изучено понятие областей эквивалентности, решены задачи на языке Haskell и представлено описание областей эквивалентности для решённых задач.

Лабораторной работа № 2  
«Исследование способов структурного тестирования программного обеспечения»

1. **Цель работы**

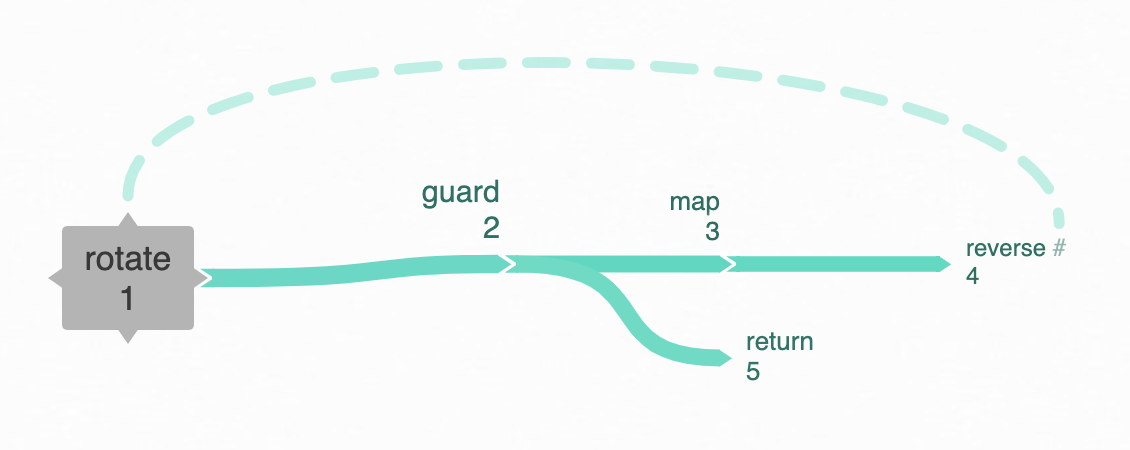
Исследовать основные подходы к структурному тестированию программного обеспечения. Приобрести практические навыки построения графа потоков управления и определения независимых ветвей программы.

1. **Постановка задачи**

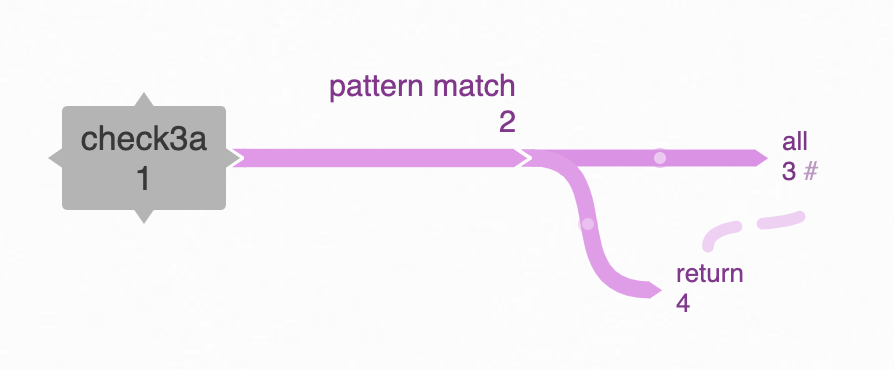
Варианты заданий соответствуют заданиям по лабораторной работе № 1. По варианту задаются требования к программам. Для каждой из них необходимо:

1. Построить граф потоков управления.
2. Вычислить цикломатическое число для построенного графа потоков управления.
3. Определить независимые ветви программы.
4. **Граф потоков управления**

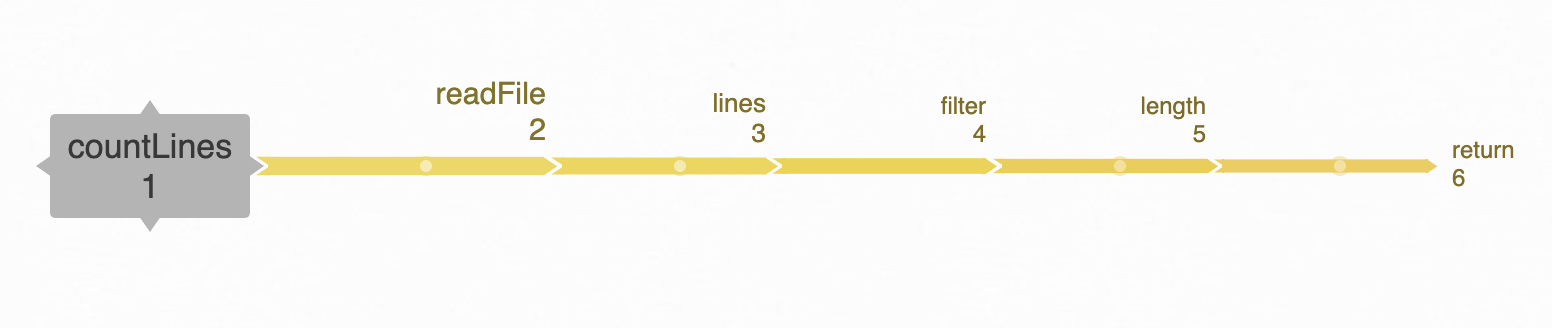
Задача 1.



Задача 2.



Задача 3.



1. **Результаты расчёта цикломатического числа**

Задача 1.

C (G) = 5 – 5 + 2 = 2

Задача 2.

C (G) = 4 – 4 + 2 = 2

Задача 3.

C (G) = 5 – 6 + 2 = 1

1. **Независимые ветви программы**

Задача 1.

Полученный граф имеет 2 независимые ветви: 1, 2, 3, 4, 1, 2, 5 и 1, 2, 5

Задача 2.

Полученный граф имеет 2 независимые ветви: 1, 2, 3 и 1, 2, 3, 4

Задача 3.

Полученный граф имеет 1 независимую ветвь: 1, 2, 3, 4, 5, 6

**Выводы**

В ходе лабораторной работы был изучен подход в тестировании на основе графа потоков управления, проведен необходимый анализ и расчёты для задач, решённых в лабораторной работе №1.

Лабораторной работа № 3  
«Исследование способов модульного тестирования программного обеспечения»

1. **Цель работы**

Исследовать основные подходы к модульному тестированию программного обеспечения. Приобрести практические навыки составления модульных тестов.

1. **Постановка задачи**
2. Выбрать в качестве тестируемого один из модулей, спроектированных в лабораторной работе №1.
3. Составить спецификацию тестового случая для одного из методов выбранного класса.
4. Реализовать тестируемый модуль и необходимое тестовое окружение.
5. Выполнить тестирование с выводом результатов на экран и сохранением в log-файл.
6. **Спецификация тестового случая**
7. Название тестируемого модуля: TMatrix
8. Название тестового случая: tMatrixTest1, tMatrixTest2
9. Описание тестового случая: тест проверяет правильность работы функции rotate – поворота матрицы на 180 градусов k раз. В первом тесте осуществляется поворот 1 раз и сравнивается с ожидаемым значением, во втором тесте проверяется, что поворот матрицы на 180 градусов 2 раза представляет собой изначальную матрицу.
10. **Программный код**

module TMatrix

( tMatrixTest1

, tMatrixTest2

) where

import Matrix (Matrix, rotate)

input :: Matrix Int

input =

[ [ 1, 2, 3, 4 ]

, [ 5, 6, 7, 8 ]

, [ 9, 0, 1, 2 ]

, [ 3, 4, 5, 6 ]

]

output :: Matrix Int

output =

[ [ 6, 5, 4, 3 ]

, [ 2, 1, 0, 9 ]

, [ 8, 7, 6, 5 ]

, [ 4, 3, 2, 1 ]

]

runTest :: (Show a, Eq a) => Matrix a -> Matrix a -> IO ()

runTest test expected = do

let content = "Expected:\n" ++ show expected ++ "\nGot:\n" ++ show test

if test == expected

then putStrLn "Passed"

else putStrLn "Failed"

writeFile "log.txt" content

tMatrixTest1, tMatrixTest2 :: IO ()

tMatrixTest1 = runTest (rotate 1 input) output

tMatrixTest2 = runTest (rotate 2 input) input

**Выводы**

В ходе лабораторной работы был изучен способ модульного тестирования программного обеспечения, а изученный материал закреплён на практике – составлена спецификация, реализовано тестовое окружение и проведено тестирование.

Лабораторной работа № 4  
«Исследование способов интеграционного тестирования программного обеспечения»

1. **Цель работы**

Исследовать основные принципы интеграционного тестирования программного обеспечения. Приобрести практические навыки организации интеграционных тестов для объектно-ориентированных программ.

1. **Постановка задачи**
2. Выбрать в качестве тестируемого взаимодействия два модуля, спроектированных в лабораторной работе №1.
3. Составить спецификацию тестового случая.
4. Реализовать тестируемый модуль и необходимое тестовое окружение.
5. Выполнить тестирование с выводом результатов на экран и сохранением в log-файл.
6. **Спецификация тестового случая**
7. Названия взаимодействующих модулей: AChecker, Matrix
8. Название теста: tMatrix3aTest1
9. Описание теста: тест проверяет, содержит ли первая строка матрицы 3 буквы а подряд, если да – поворачивает матрицу на 180 градусов 1 раз, если нет – возвращает исходную матрицу.
10. Начальные условия: на вход поступает матрица “aaa”, “bbb”, “ccc”
11. Ожидаемый результат: матрица повёрнута на 180 градусов, “ccc”, “bbb”, “aaa”
12. **Программный код**

module TMatrix3a

( tMatrix3aTest1

) where

import AChecker (check3a)

import Matrix (Matrix, rotate)

input :: Matrix Char

input =

[ "aaa"

, "bbb"

, "ccc"

]

output :: Matrix Char

output =

[ "ccc"

, "bbb"

, "aaa"

]

runTest :: Matrix Char -> Matrix Char -> IO ()

runTest test expected = do

let content = "Expected:\n" ++ show expected ++ "\nGot:\n" ++ show test

if test == expected

then putStrLn "Passed"

else putStrLn "Failed"

writeFile "log.txt" content

tMatrix3aTest1 :: IO ()

tMatrix3aTest1 = runTest test output

where test = if check3a $ head input then rotate 1 input else input

**Выводы**

В ходе лабораторной работы был изучен способ интеграционного тестирования программного обеспечения, а изученный материал закреплён на практике – составлена спецификация, реализовано тестовое окружение и проведено тестирование.

Лабораторной работа № 5  
«Исследование способов модульного тестирования программного обеспечения в среде HUnit»

1. **Цель работы**

Исследовать эффективность использования методологии TDD при разработке программного обеспечения. Получить практические навыки использования фреймворка HUnit для модульного тестирования программного обеспечения.

1. **Постановка задачи**

Необходимо разработать и запустить набор тестов, проанализировать результаты. При этом необходимо использовать библиотеку HUnit.

1. **Описание обязанностей тестируемого модуля**

Тестируемый модуль отвечает за корректность указанной ftp ссылки, проверяет доступен внешний ресурс.

1. **Программный код**

Программный код доступен по ссылке:

<https://github.com/serokell/xrefcheck/blob/master/links-tests/Test/Xrefcheck/FtpLinks.hs>

**Выводы**

В ходе лабораторной работы были изучены возможности модульного тестирования при помощи библиотеки HUnit.

Лабораторной работа № 6  
«Исследование способов профилирования программного обеспечения»

1. **Цель работы**

Исследовать критические по времени выполнения участки программного кода и возможности их устранения. Приобрести практические навыки анализа программ с помощью профайлера.

1. **Постановка задачи**
2. Разработать программу для исследования.
3. Выполнить профилирования разработанной программы, выявить функции, на выполнение которых тратится наибольшее время.
4. Модифицировать программу с целью оптимизации времени выполнения.
5. Выполнить повторное профилирование программы, сравнить новые результаты и полученные ранее, сделать выводы.
6. **Программный код**

module Main where

myConcat :: [[a]] -> [a]

myConcat = {-# SCC "concat" #-} foldl (++) []

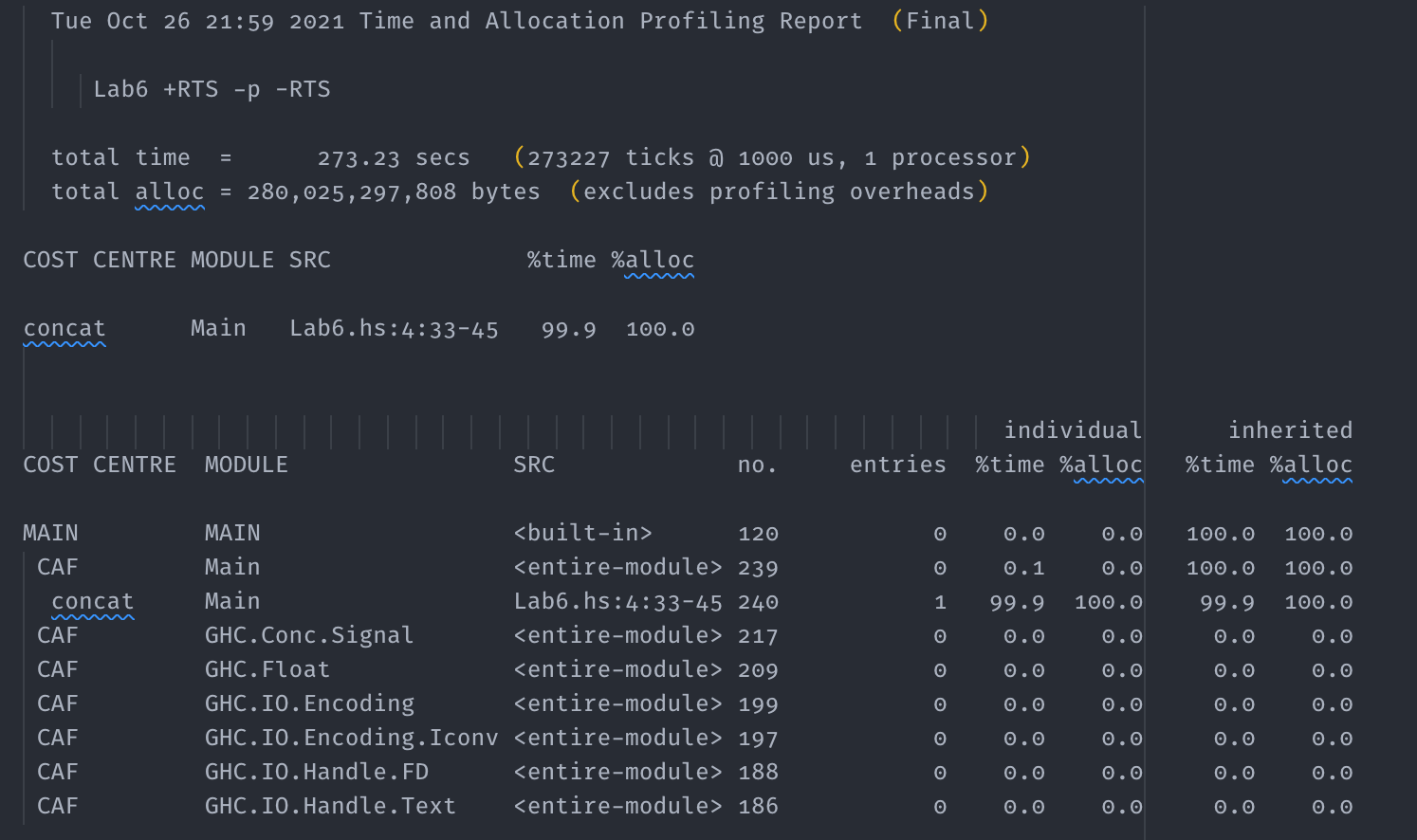
test :: Double

test = last . myConcat $ map return [1 .. 1e5]

main :: IO ()

main = print test

1. **Результаты профилирования**



1. **Модифицированный программный код**

module Main where

myConcat :: [[a]] -> [a]

myConcat = {-# SCC "concat" #-} foldr (++) []

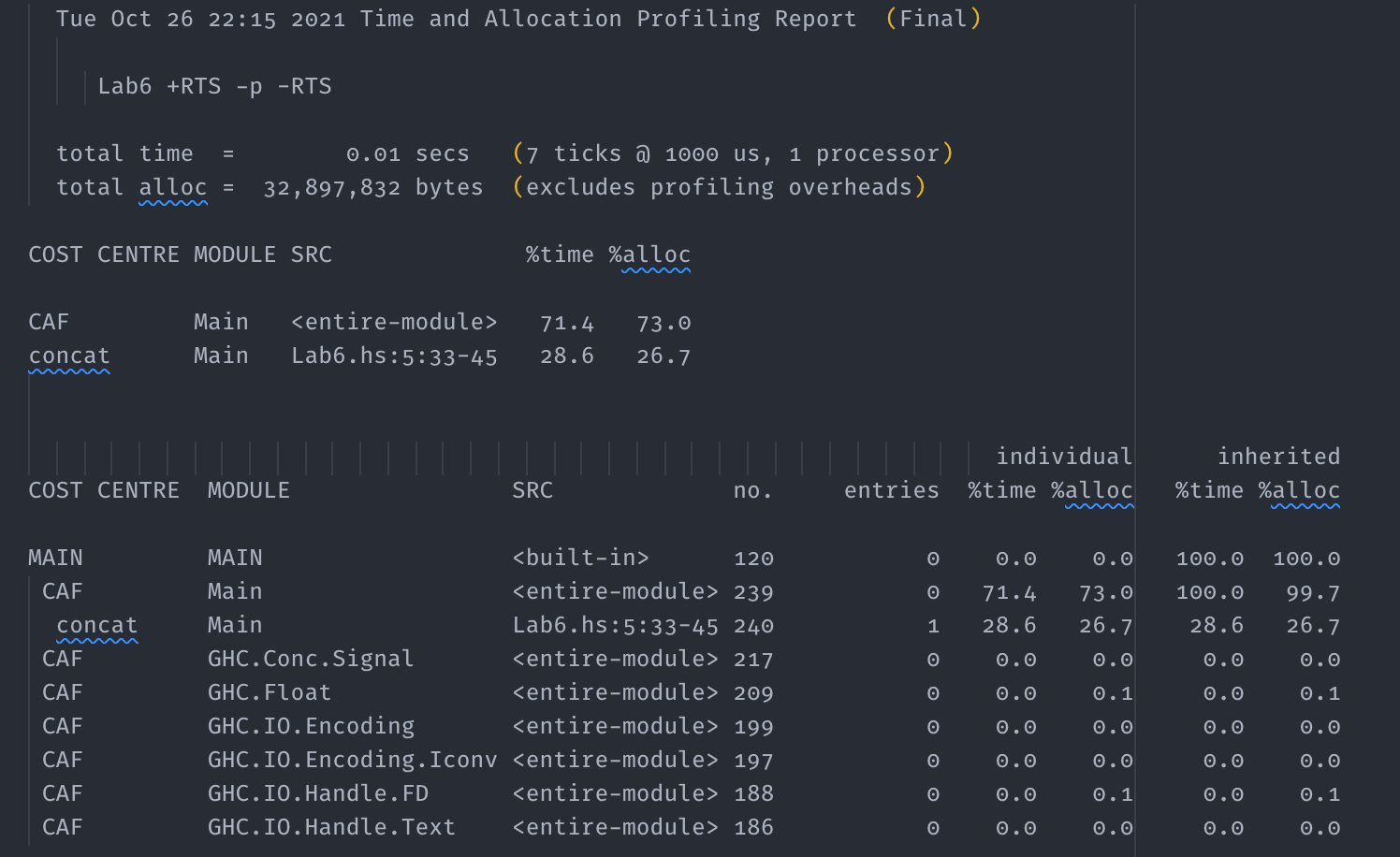
test :: Double

test = last . myConcat $ map return [1 .. 1e5]

main :: IO ()

main = print test

1. **Результаты профилирования**



**Выводы**

В ходе лабораторной работы были изучены возможности профилирования, на примере левой и правой свёртки мы увидели насколько значительно может отличаться время выполнения программы.