|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК5 «Системы обработки информации»** |

**ЛАБОРАТОРНАЯ РАБОТА №1**

**«**Инструментальные средства, основания языка Haskell**»**

**ДИСЦИПЛИНА:** «Перспективные технологии разработки программных средств»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-31М | |  |  | ( | Сафронов Н.С, | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Кириллов В.Ю. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

Калуга, 2025

**Цель:**

Формирование практических навыков подготовки и настройки инструментальной среды, написания и анализа элементарных программ на функциональном языке программирования, адаптация к функциональной парадигме.

**Задачи:**

• Подготовка и настройка платформы для Haskell.

• Программирование и анализ результатов.

• Написание отчета о работе.

**Результаты выполнения работы**

**Вопрос 2.1**

Для написания функции calcChange мы использовали конструкцию if then else. В Haskell все if-выражения должны содержать компонент else. Почему, согласно нашим трём правилам для функций, if нельзя использовать само по себе?

**Ответ:**

В Haskell нет «ничего» или «неопределённого поведения» как в императивных языках. Если бы `if` мог существовать без `else`, то

при ложном условии выражение не возвращало бы ничего — что нарушает принцип тотальности функций.

Каждое выражение в Haskell имеет строго определённый тип. Если бы else отсутствовал, то при ложном условии тип выражения стал

бы неопределённым (или undefined), что нарушает систему типов и принцип детерминированности.

В императивных языках if без else может просто «ничего не делать» — но это подразумевает побочный эффект (отсутствие действия).

В Haskell нет «ничего не делать» — каждая ветвь должна явно возвращать значение, чтобы сохранить чистоту и предсказуемость.

**Задание 7.2**

-- | myGCD выполняет поиск наибольшего общего делителя.

-- Пример использования: `myGCD 100 32` -> 4

myGCD a b = case a `mod` b of

    0 -> b

    r -> myGCD b r

**Задание 8.1**

-- | myReverse возвращает список с инвертированным порядком следования объектов.

-- Пример использования: `myReverse [1..5]` -> `[5,4,3,2,1]`

myReverse (x:xs) = case length xs of

0 -> [x]

r -> myReverse xs ++ [x]

**Задание 8.2**

-- | fastFib реализация быстрого подсчёта чисел Фибоначчи.Applicative

-- Пример использования: `fastFib 1 1 5` -> 8

fastFib n1 n2 1 = n2

fastFib n1 n2 counter = fastFib n2 (n1 + n2) (counter - 1)

**Задание 10.2**

{-# LANGUAGE ImpredicativeTypes #-}

{-# LANGUAGE RankNTypes #-}

type Robot = forall a. ((String, Int, Int) -> a) -> a

-- | Конструктор робота: принимает тройку (имя, атака, здоровье)

-- и возвращает "робота" — функцию, которая принимает сообщение (функцию)

-- и применяет его к внутреннему состоянию.

robot :: (String, Int, Int) -> Robot

robot (name, attack, hp) = \message ->

message (name, attack, hp)

name :: (String, a, b) -> String

name (n, \_, \_) = n

attack :: (a, Int, b) -> Int

attack (\_, a, \_) = a

hp :: (a, b, Int) -> Int

hp (\_, \_, hp) = hp

getName :: Robot -> String

getName aRobot = aRobot name

getAttack :: Robot -> Int

getAttack aRobot = aRobot attack

getHP :: Robot -> Int

getHP aRobot = aRobot hp

setName :: Robot -> String -> Robot

setName aRobot newName =

aRobot

( \(n, a, h) ->

robot (newName, a, h)

)

setAttack :: Robot -> Int -> Robot

setAttack aRobot newAttack =

aRobot

( \(n, a, h) ->

robot (n, newAttack, h)

)

setHP :: Robot -> Int -> Robot

setHP aRobot newHP =

aRobot

( \(n, a, h) ->

robot (n, a, newHP)

)

printRobot :: Robot -> String

printRobot aRobot =

aRobot

( \(n, a, h) ->

n

++ " attack HP:"

++ show a

++ " HP:"

++ show h

)

damage :: Robot -> Int -> Robot

damage aRobot attackDamage =

aRobot (\(n, a, h) -> robot (n, a, h - attackDamage))

fight :: Robot -> Robot -> Robot

fight aRobot defender = damage defender attack

where

attack =

if getHP aRobot > 10

then getAttack aRobot

else 0

-- Далее идут дополнения из пункта "Расширение проекта"

-- Задание 1

-- | Используя map на списке роботов, получите количество жизни каждого робота в списке

getHPs :: [Robot] -> [Int]

getHPs robots = map getHP robots

-- Задание 2

-- | getFightWinner - функция с аннотацией типа, чтобы было возможно реализовать перестановки аргументов

getFightWinner :: Robot -> Robot -> Robot

getFightWinner robotA robotB =

if getHP robotA > getHP robotB

then robotA

else robotB

-- | Функция multiroundFight, принимающая на вход двух роботов, заставляющую их драться в течение N раундов и

-- возвращающую победителя

multiroundFight :: Robot -> Robot -> Int -> Robot

multiroundFight robotA robotB count =

case count of

1 ->

getFightWinner

(fight robotB robotA)

(fight robotA robotB)

r ->

multiroundFight

(fight robotB robotA)

(fight robotA robotB)

(count - 1)

{-

Сценарий битвы:

robotA = robot("RA", 30, 90)

robotB = robot("RB", 30, 100)

printRobot (multiroundFight robotA robotB 3)

-}

-- Задание 3

robot1 :: Robot

robot1 = robot ("R1", 10, 50)

robot2 :: Robot

robot2 = robot ("R2", 12, 45)

robot3 :: Robot

robot3 = robot ("R3", 8, 60)

robots :: [Robot]

robots = [robot1, robot2, robot3]

boss :: Robot

boss = robot ("Boss", 20, 100)

fightBoss :: Robot -> Robot

fightBoss = fight boss

damagedRobots :: [Robot]

damagedRobots = map fightBoss robots

remainingHPs :: [Int]

remainingHPs = map getHP damagedRobots

**Вывод:** в процессе выполнения лабораторной работы сформированы практических навыков подготовки и настройки инструментальной среды, написания и анализа элементарных программ на функциональном языке программирования, адаптация к функциональной парадигме.