**Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования**

**Московский технический университет связи и информатики**

**КАФЕДРА ИНФОРМАТИКИ**

**Лабораторная работа по теме 3  
Работа с дополнительными функциями и классами Haskell**

**Выполнил:**

**Студент группы БВТ1701**

**Шабанов Борис**

**Вариант 22**

**Москва 2019**

**Цель работы:** изучить дополнительные функции и классы языка Haskell, использовать в дальнейшем для написания программы.

**1 Формулировка задания на лабораторную работу**

Описать функцию, которая находила бы сумму всех числовых элементов списка. Список может содержать подсписки произвольной глубины.

**2 Техническое задание**

-Параметры ввода: a = список

-Параметры вывода: sum = типа Int – сумма всех элементов.

**3 Функции программы**

func1 a = sum a – функция, высчитывающая сумма элементов списка.

func2 a = sum (map sum a) – функция, высчитывающая сумму элементов подсписков.

1. **Код программы**

func1 a = sum a

func2 a = sum (map sum a)

1. **Результат программы**

На рисунке 1 изображен итог программы.

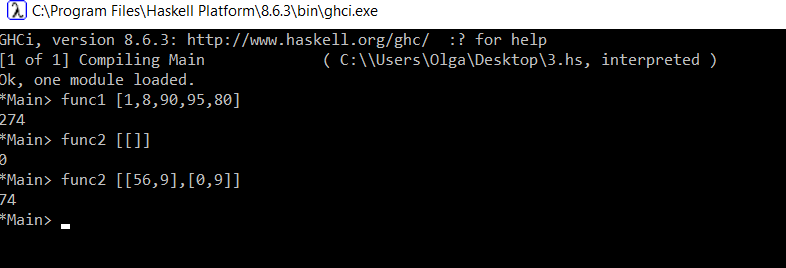


Рисунок 2 - результаты программы.

1. **Теоретическая часть**

Значение типа **Maybe** a представляет значение типа a, но с прикреплённым контекстом возможной неудачи в вычислениях. Значение **Just** "дхарма" означает, что в нём имеется строка "дхарма". Значение **Nothing** представляет отсутствие значения, или, если вы посмотрите на строку как на результат вычисления, это говорит о том, что вычисление завершилось неуспешно.

Когда мы рассматривали тип Maybe как функтор, мы видели, что если нам нужно отобразить его с помощью функции, используя метод fmap, функция отображала содержимое, если это значение Just. В противном случае cохранялось значение Nothing, поскольку с помощью функции нечего отображать!

ghci> fmap (++"!") (Just "мудрость")

Just "мудрость!"

ghci> fmap (++"!") Nothing

Nothing

Ещё один чудесный тип, принимающий два других в качестве параметров, – это тип **Either**. Он определён приблизительно так: data Either a b = Left a | Right b deriving (Eq, Ord, Read, Show) У него два конструктора данных. Если используется конструктор Left, его содержимое имеет тип а; если Right – содержимое имеет тип b. Таким образом, мы можем использовать данный тип для инкапсуляции значения одного из двух типов. Когда мы работаем

с типом Either a b, то обычно используем сопоставление с образцом по Left и Right и выполняем действия в зависимости от того, какой вариант совпал.

ghci> Right 20

Right 20

ghci> Left "в00т"

Left "в00т"

ghci> :t Right 'a'

Right 'a' :: Either a Char

ghci> :t Left True

Left True :: Either Bool b

Из приведённого примера следует, что типом значения Left True является Either Bool b. Первый параметр типа Bool, поскольку значение создано конструктором Left; второй же параметр остался полиморфным. Ситуация подобна тому как значение Nothing имеет тип Maybe a

**Reader**:

Выражение **do** всегда возвращает монадическое значение, и данное выражение ничем от него не отличается. Результатом этого монадического значения является функция. Она принимает число, затем к этому числу применяется функция (\*2) и результат записывается в образец a. К тому же самому числу, к которому применялась функция (\*2), применяется теперь уже функция (+10), и результат записывается в образец b. Функция **return,**

как и в других монадах, не имеет никакого другого эффекта, кроме создания монадического значения, возвращающего некий результат. Она возвращает значение выражения (a+b) в качестве результата данной функции. Если мы протестируем её, то получим те же результаты, что и прежде:

ghci> addStuff 3

19

И функция (\*2), и функция (+10) применяются в данном случае к числу 3. Выражение return (a+b) применяется тоже, но оно игнорирует это значение и всегда возвращает (a+b) в качестве результата. По этой причине функциональную монаду также называют монадой-читателем. Все функции читают из общего источника

Монада **Writer** позволяет нам производить вычисления, в то же время обеспечивая слияние всех журнальных значений в одно, которое затем присоединяется к результату. Например, мы могли бы снабдить наши значения строками, которые объясняют, что происходит, возможно, для отладочных целей. Рассмотрите функцию, которая принимает число бандитов в банде и сообщает нам, является ли эта банда крупной. Это очень

простая функция:

isBigGang :: Int –> Bool

isBigGang x = x > 9