**Федеральное агентство связи**

**Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования «Московского технического университета связей и информатики»**

Кафедра МКиТИ

Лабораторная работа №2

Выполнила:

Студентка группы БВТ1702

Перевозова Ю.В.

Вариант 21

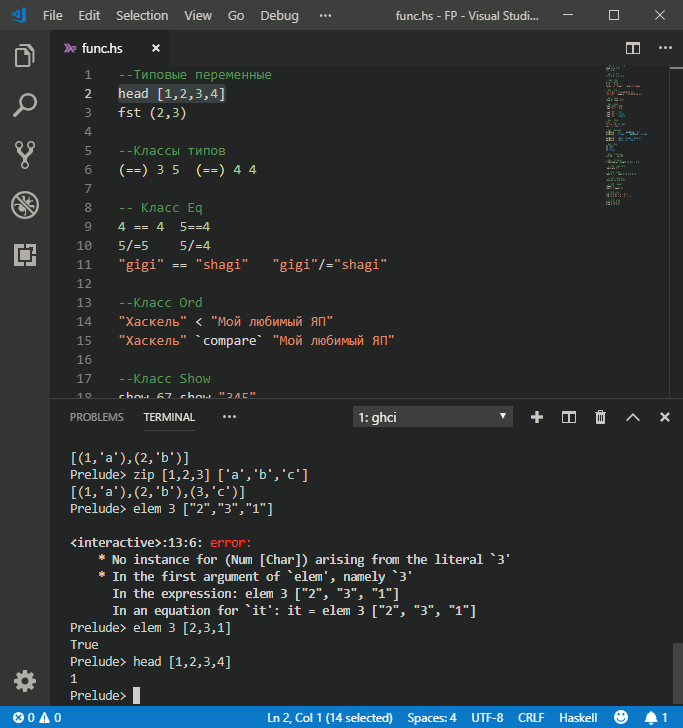
Москва 2019

**Цель работы:** изучить основные типы и классы типов, а также некоторые рекурсивные функции в Haskell.

**Типовые переменные**

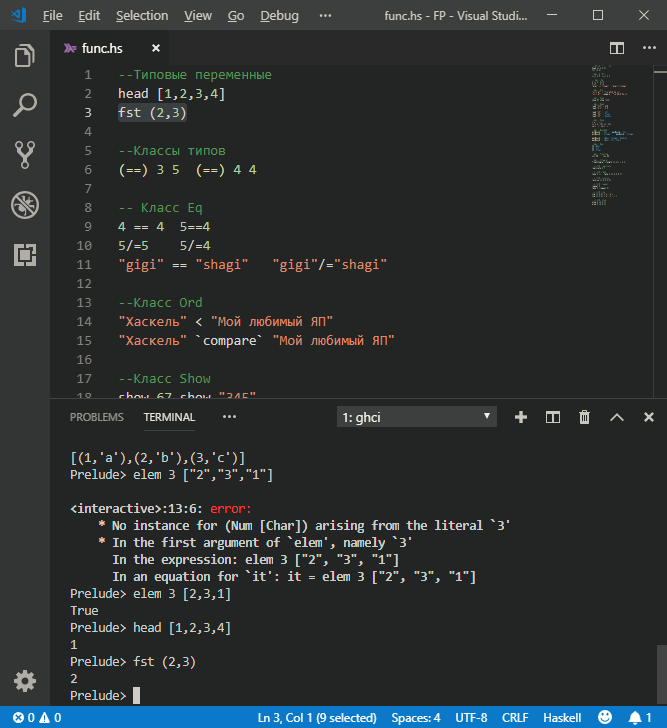
В Haskell есть так называемые типовые переменные. Мы просто вместо конкретного типа указываем его псевдоним. Псевдоним для типовой переменной может быть любой строкой, начинающейся со строчной буквы. Но чаще всего их называют одной-двумя буквами. Например функция head, которая возвращает головной элемент списка имеет следующую сигнатуру: head :: [a] -> a. Она принимает список элементов любого типа и возвращает одно значение такого же типа.

* 1. head [1,2,3,4]



Важно знать, что переменные типа с разными именами не обязательно должны иметь значения разных типов. Например функция fst, которая возвращает первый элемент кортежа имеет сигнатуру fst :: (a, b) -> a.

* 1. fst (2,3)



**Класс эквивалентности (Eq)**

Класс Eq используется для типов, поддерживающих сравнение на равенство. Методы класса:

(==) :: a -> a -> Bool infix 4

(/=) :: a -> a -> Bool

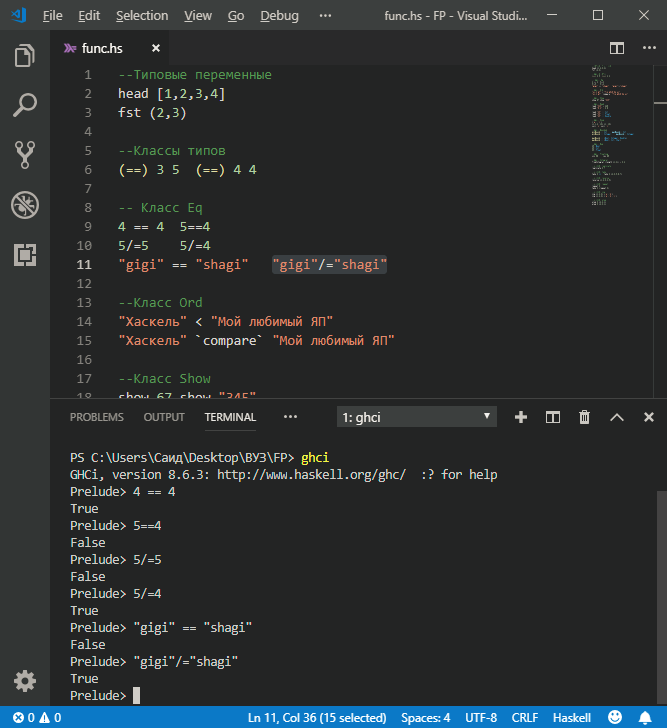
Его экземпляры должны реализовать по крайней мере один из этих методов.

**Примеры:**

4 == 4 5==4

5/=5 5/=4

"gigi" == "shagi" "gigi"/="shagi"



**Класс Ord**

Класс Ord используется для упорядоченных типов. Методы класса:

(<) :: a -> a -> Bool

(<=) :: a -> a -> Bool

(>) :: a -> a -> Bool

(>=) :: a -> a -> Bool

compare :: a -> a -> Ordering -- Аналог оператора Comparison в C#.

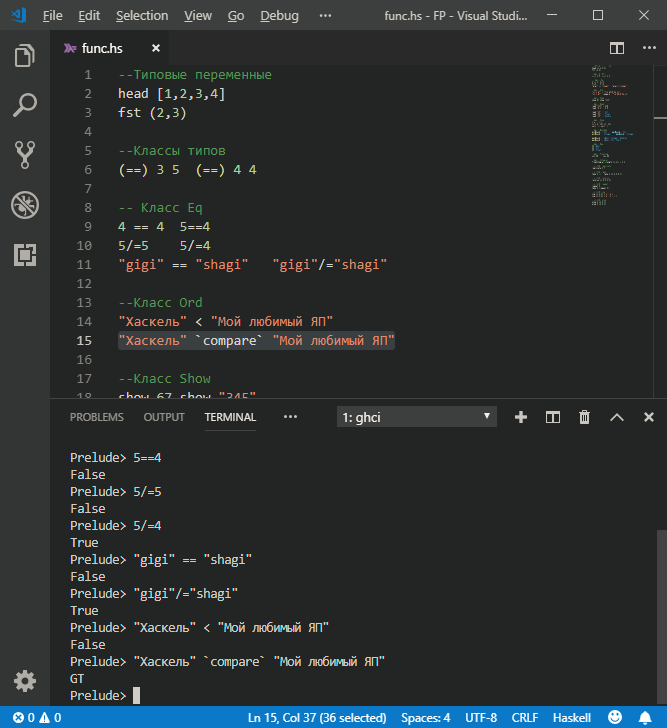
max :: a -> a -> a

min :: a -> a -> a

**Примеры:**

"Хаскель" < "Мой любимый ЯП"

"Хаскель" `compare` "Мой любимый ЯП"



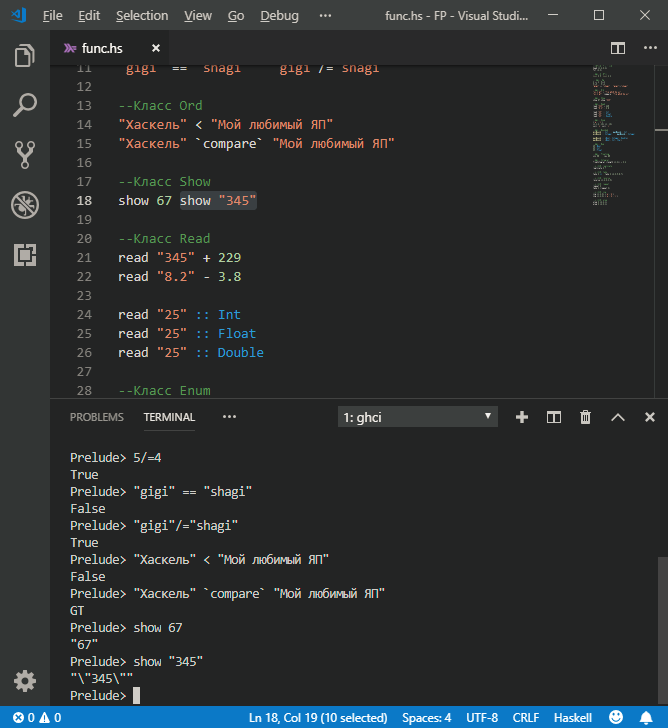
**Класс Show**

Класс для типов, значения которых можно преобразовать в строку или можно представить как символы.

**Примеры:**

show 67

show "345"



**Класс Read**

Класс для типов, значения которых можно преобразовать из строки.

**Примеры:**

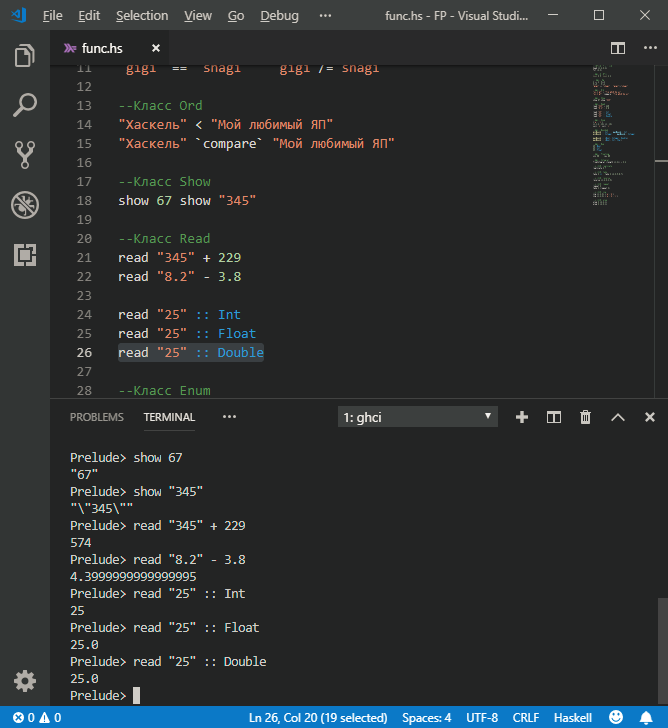
read "345" + 229

read "8.2" - 3.8

read "25" :: Int

read "25" :: Float

read "25" :: Double



**Класс Enum**

Класс Enum используется для упорядоченных последовательностей. Типы этого класса можно использовать в интервалах, например: ['a'..'z'].

**Методы класса**:

succ :: a -> a -- Возвращает следующий элемент.

pred :: a -> a -- Возвращает предыдущий элемент.

toEnum :: Int -> a -- Переводит значение типа Int в значение типа Enum.

fromEnum :: a -> Int -- Переводит значение типа Enum в значение типа Int.

enumFrom :: a -> [a]

enumFromThen :: a -> a -> [a]

enumFromTo :: a -> a -> [a]

enumFromThenTo :: a -> a -> a -> [a]

**Примеры:**

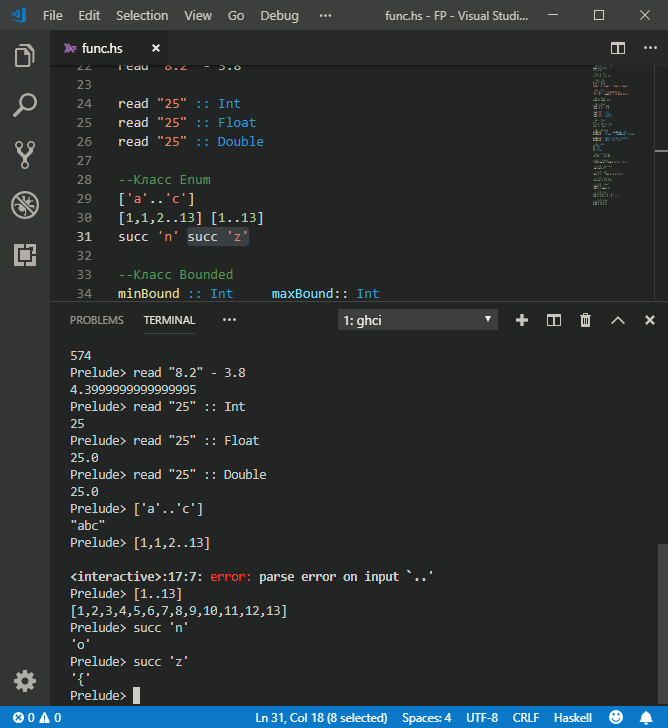
['a'..'c']

[1,1,2..13]

[1..13]

succ 'n'

succ 'z'



**Класс Bounded**

Класс Bounded используется для типов у которых есть минальное и максимальное допустимые значения. Например тип Int имеет максимальное и минимальное значения.

Этот класс содержит два метода, которые обязательны для реализации: minBound и maxBound, возвращающие минимальное и максимальное значения соответственно.

**Примеры:**

minBound :: Int

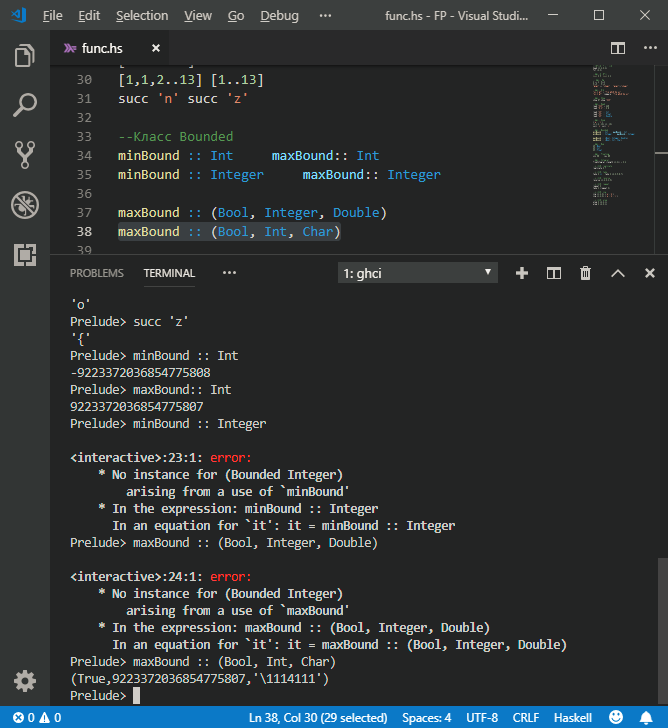
maxBound:: Int

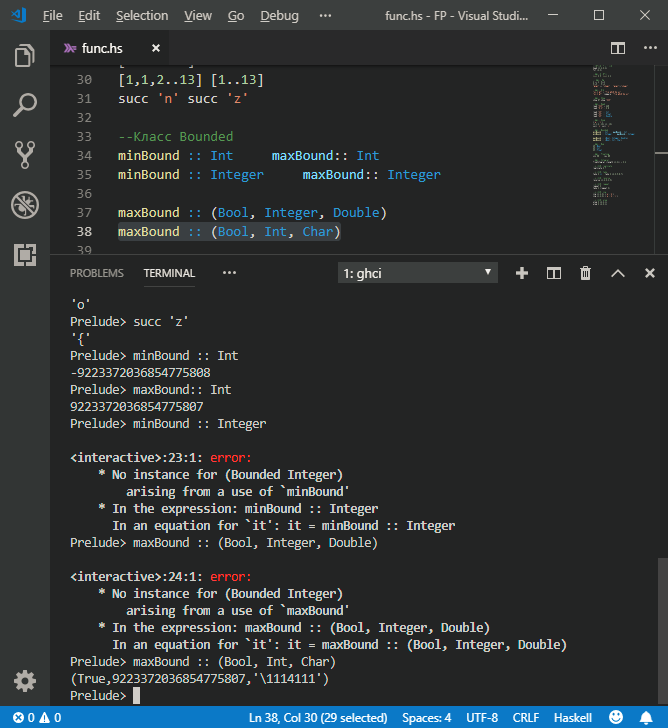
minBound :: Integer

maxBound:: Integer

maxBound :: (Bool, Integer, Double)

maxBound :: (Bool, Int, Char)





**Класс Num**

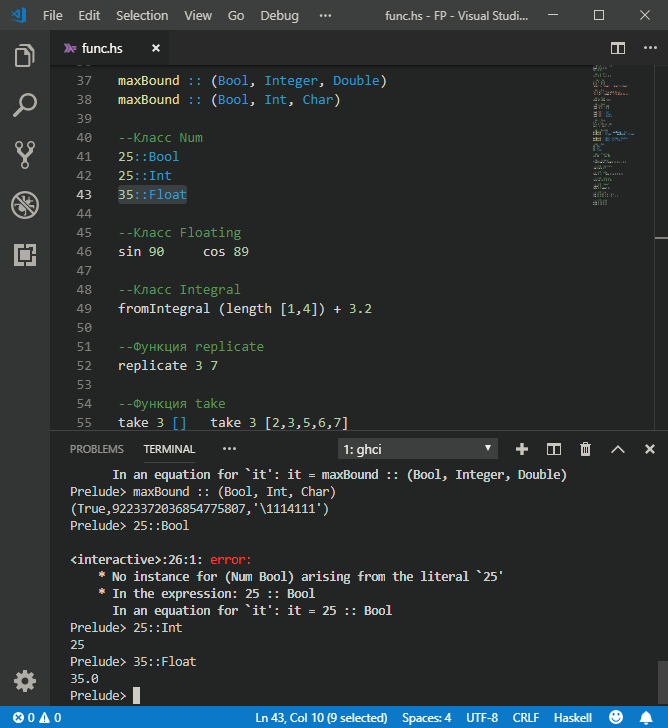
Num – класс чисел. Представители: Double, Float, Int, Integral.

**Примеры:**

25::Bool

25::Int

35::Float



**Класс Floating**

В данный класс входят числа с плавающей точкой одинарной точности.

**Представители:** Double, Float.

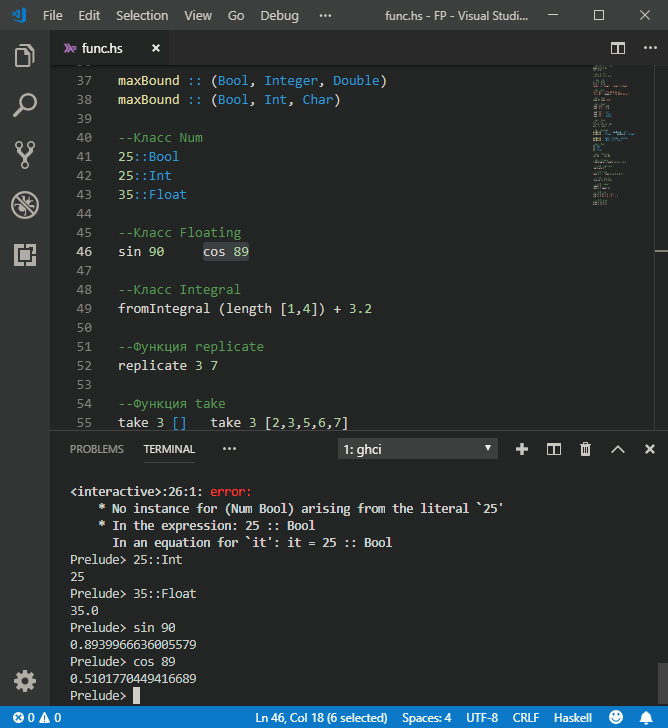
**Методы:**

* Квадратный корень sqrt :: a -> a
* Возведение в степень (\*\*) :: a -> a -> a
* Число π = 3.1415926…: pi :: a
* Экспонента exp :: a -> a и натуральный логарифм log :: a -> a
* Логарифм по данному основанию logBase :: a -> a -> a
* Тригонометрические функции типа a -> a: sin, tan, cos, asin, atan, acos, sinh, tanh, cosh, asinh, atanh, acosh.

**Примеры:**

sin 90

cos 89



**Класс Integral**

Integral – класс целых чисел.

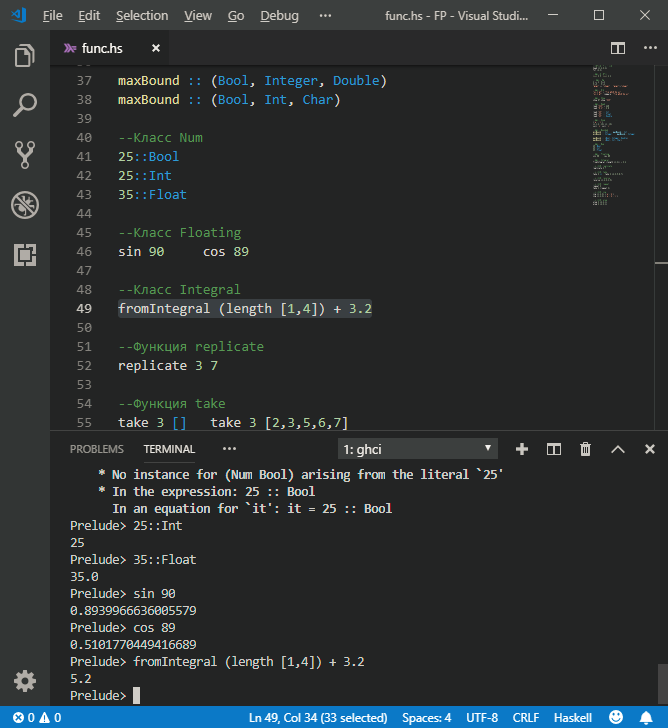
**Представители**: Int, Integral.

**Методы**:

* Целочисленное деление с округлением к 0: quot :: a -> a -> a
* Остаток от деления quot: rem :: a -> a -> a
* Целочисленное деление с округлением к −∞: div :: a -> a -> a
* Остаток от деления div: mod :: a -> a -> a
* Пара чисел (частное, остаток): quotRem :: a -> a -> (a, a), divMod :: a -> a -> (a, a)
* Преобразование в целое число типа Integer: toInteger :: a -> Integer

**Примеры**:

fromIntegral (length [1,4]) + 3.2

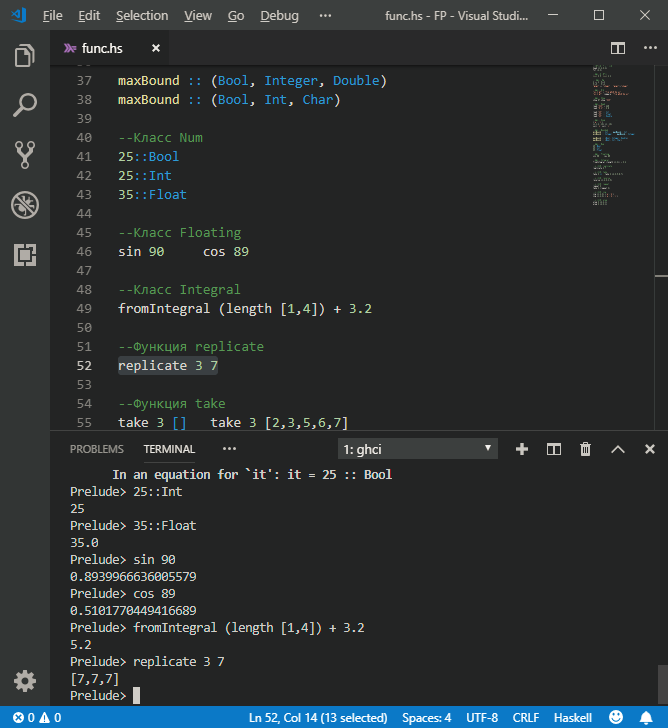


**Функция Replicate**

Cоздает список длины, заданной первым аргументом, и элементов, имеющих значение второго аргумента.

**Примеры:**

replicate 3 7



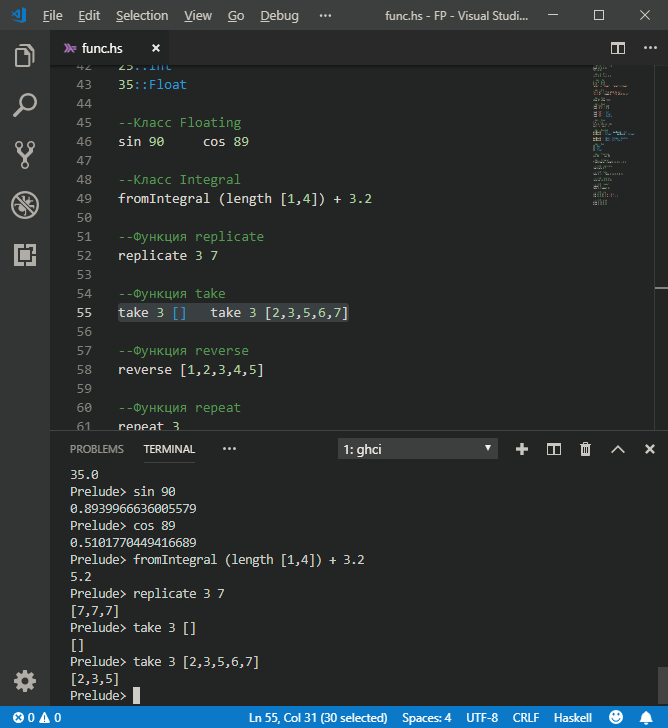
**Функция take**

Данная функция создает список, первый аргумент которой определяет, сколько элементов должно быть взято из списка, переданного в качестве второго аргумента.

**Примеры:**

take 3 []

take 3 [2,3,5,6,7]

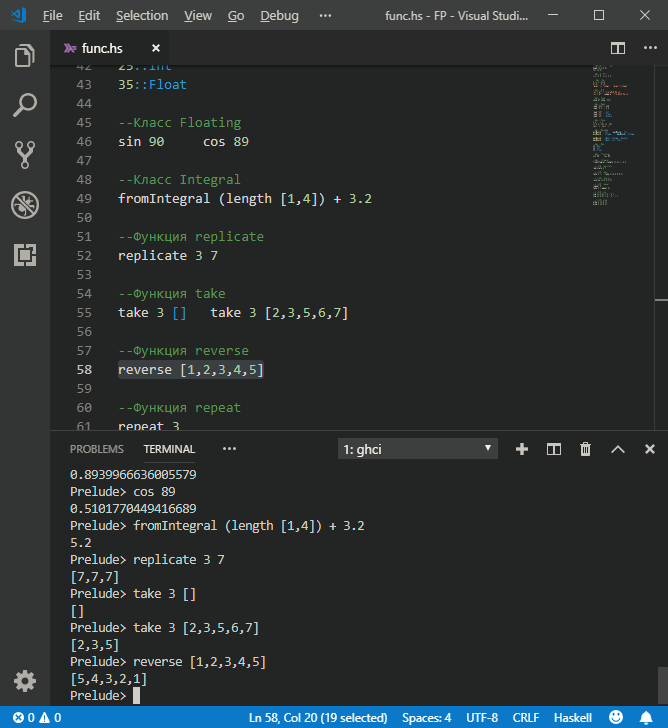
****

**Функция reverse**

Cоздает новую строку из исходной строки с элементами в обратном порядке.

**Примеры:**

reverse [1,2,3,4,5]



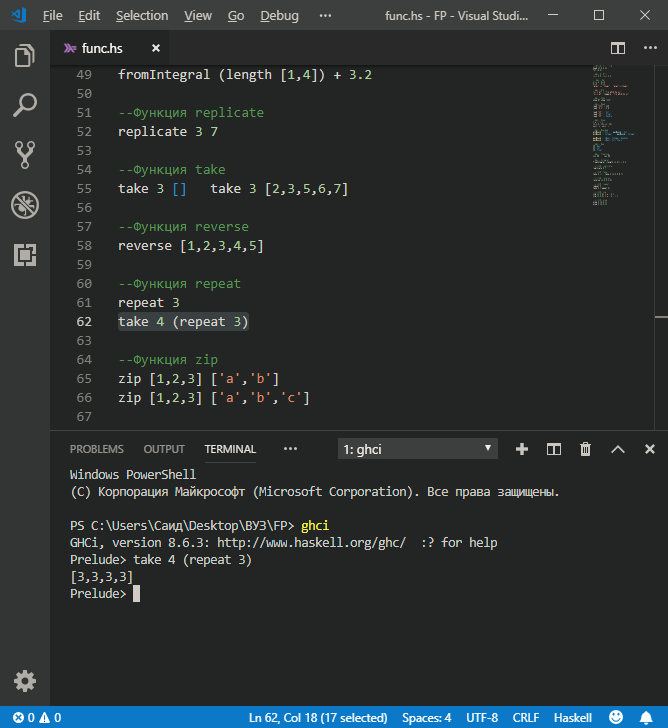
**Функция Repeat**

Cоздает бесконечный список, где все элементы дублируется согласно введенному первому элементу.

**Примеры:**

repeat 3

take 4 (repeat 3)



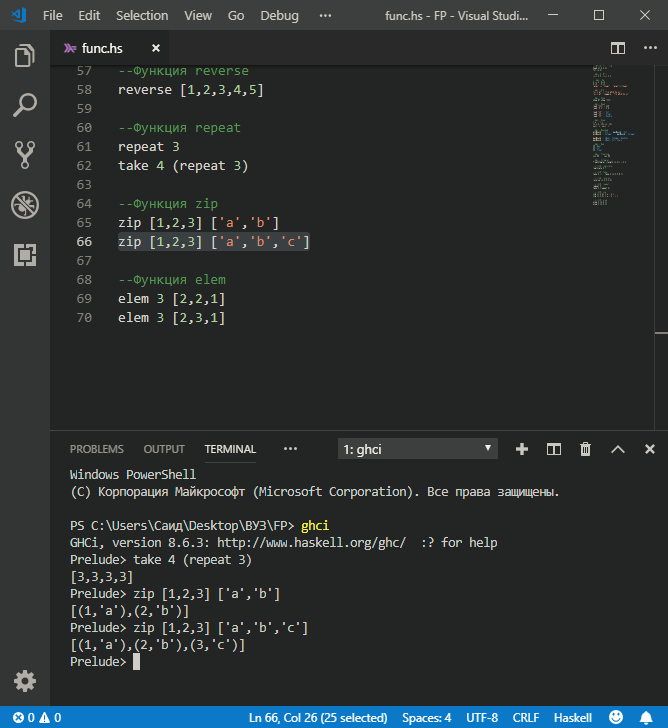
**Функция Zip**

Данная функция создает список кортежей, каждый кортеж которой содержит элементы обоих списков, находящиеся в одной и той же позиции.

**Примеры:**

zip [1,2,3] ['a','b']

zip [1,2,3] ['a','b','c']



**Функция Elem**

Возвращает True, если список содержит элемент, равный первому аргументу, и false если условие не выполняется.

**Примеры:**

elem 3 [2,2,1]

elem 3 [2,3,1]

