# Лабораторная работа 5, задание 8.2 (по выбору)

## Условие

Добавить к заданию 8.2 методы ввода/вывода с файла, а также ввод/вывод с консоли во время исполнения программы

Условия 8.2

Иерархия должностей в некоторой организации образует древовидную структуру. Каждый работник, однозначно характеризующийся уникальным именем, имеет несколько подчиненных. Определите тип данных, представляющий такую иерархию и опишите следующие функции:

2. getAllSubordinate, возвращающую список всех подчиненных данного работника, включая косвенных.

## Код

## {-

## ПИ19-3 Деменчук

## Задание 8.2 (по выбору)

## 8. Иерархия должностей в некоторой организации образует древовидную структуру.

## Каждый работник, однозначно характеризующийся уникальным именем, имеет несколько подчиненных.

## Определите тип данных, представляющий такую иерархию и опишите следующие функции:

## 2) getAllSubordinate, возвращающую список всех подчиненных данного работника, включая косвенных.

## -}

## -- Создаем кастомный тип, согласно заданию

## -- у каждого работника есть массив подчиненных

## data Employee = JobWorker (String) [Employee] deriving (Show, Eq, Read)

## -- Метод по поиску значения

## find ((employee, emp) :xs) manager =

## if employee == manager then emp

## else find xs manager

## -- Список имён работников

## workerNamesList (JobWorker (n) emp) = n:concatMap workerNamesList emp

## -- Список всех работников

## employeeList (JobWorker (name) emp) = (name, emp):concatMap employeeList emp

## -- getAllSubordinate, возвращает список всех подчиненных работника

## getAllSubordinate employee (JobWorker (name) emp) =

## concat (map (\item -> workerNamesList item) (find (employeeList

## (JobWorker (name) emp)) employee))

## -- Ввод данных

## readData :: String -> IO String

## readData (userInput) = if (userInput == "1") then getLine else readFile "word.txt"

## --Сохранение струкутры в файл

## save :: Employee -> FilePath -> IO ()

## save editor f = writeFile f $ show editor

## -- Конвертер для преобразования String в массив Char

## myconverter :: String -> [Char]

## myconverter = id

## -- Точка входа в приложение

## main = do

## --Читаем наше дерево

## buf <- readFile "tree.txt"

## let tree = read buf :: Employee

## -- Спрашиваем пользователя о том, как он хочет ввести данные

## putStrLn "Как считать данные о работнике?\n1. С клавиатуры\n2. С файла"

## userInput <- getLine

## -- Вызов readData

## input <- readData userInput

## putStrLn ("Ввод: " ++ input)

## -- Преобразование в массив char

## let converted = myconverter input

## -- Получение иерархии от getAllSubordinate

## let result = getAllSubordinate converted tree

## -- Спрашиваем как выводить данные будем

## putStrLn "Как вывести данные?\n1. На экран\n2. В файл"

## userInput <- getLine

## -- Запись ворматированного вывода в переменную

## let formatedResult = "Результат: " ++ show result

## -- Если вывод на экран

## if userInput == "1" then do

## putStrLn formatedResult

## -- Если вывод в файл

## else do

## writeFile "result.txt" formatedResult

## putStrLn "Занесли данные в файл"

## save tree "tree.txt"

## {-

## -- Формируем тестовую иерархию работников 1

## print(getAllSubordinate "Ivanov" (

## JobWorker "Ivanov"[

## JobWorker "Petrov"

## [

## JobWorker "Sidorov" [],

## JobWorker "Demko" [],

## JobWorker "Dmitriev" []

## ],

## JobWorker "Maximov" []

## ]

## ))

## -- Представляем иерархию работников в виде чисел для наглядности

## print(getAllSubordinate "Petrov" (

## JobWorker "Petrov" [JobWorker "1"

## [JobWorker "1.1" [],

## JobWorker "1.2" [],

## JobWorker "1.3" []

## ],

## JobWorker "2"

## [JobWorker "2.1" [

## JobWorker "2.1.1" []

## ],

## JobWorker "2.2" [],

## JobWorker "2.3" []

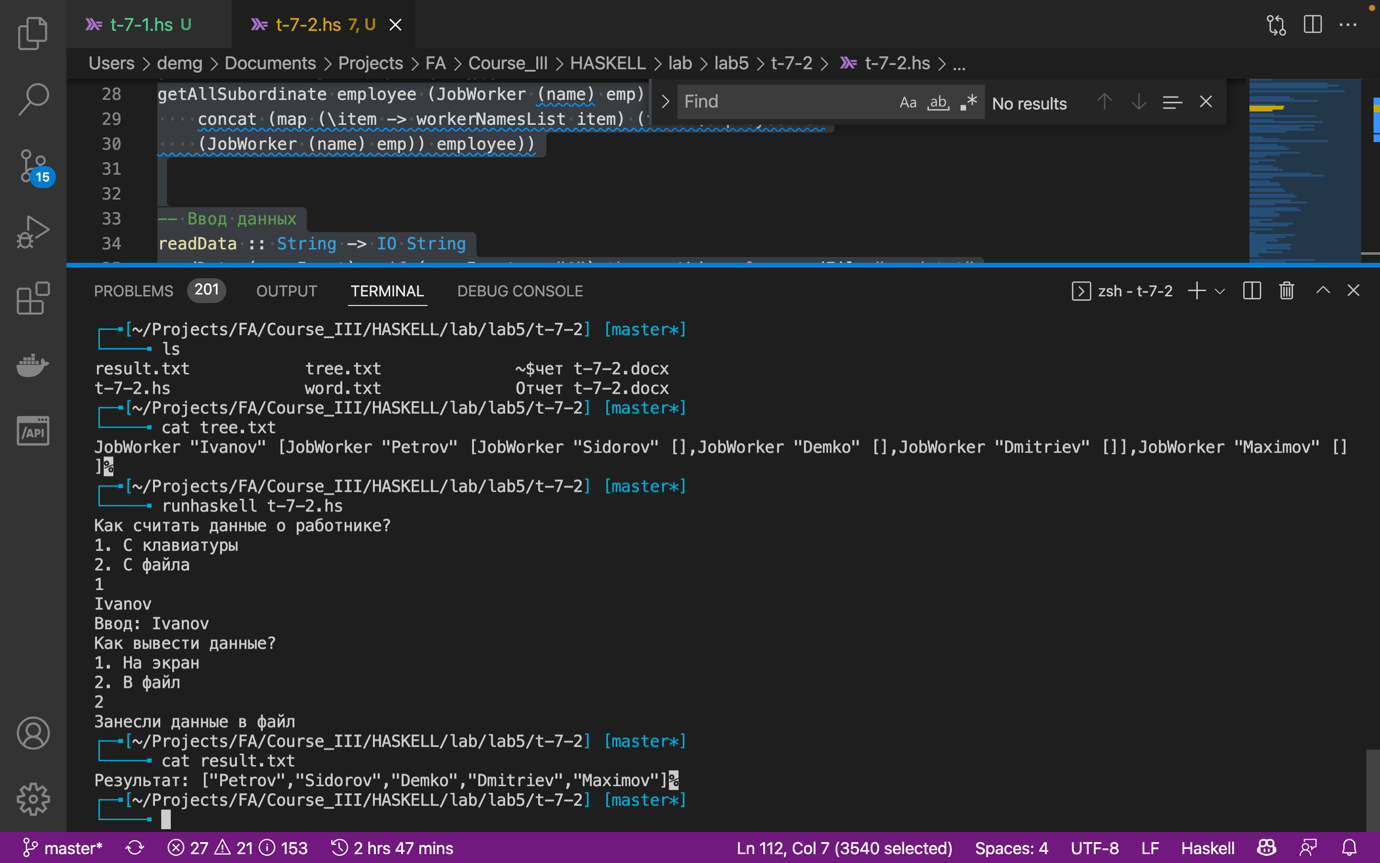
## ]

## ]))

## -}

## Скриншоты

Пример ввода с консоли и выводом в файл result.txt



Пример ввода с файла и отображением на консоли

