Основната цел на проекта е създаването на програма, която симулира файлова система, реализирана на Haskell. Тази система предлага функционалности за създаване, управление, навигиране и визуализиране на файлове и папки в йерархична структура. Допълнително, програмата интегрира персонализиран парсер за обработка на потребителски команди, предоставяйки интуитивно взаимодействие, наподобяващо Shell интерфейс, за ефективно управление на файловата система.

1. Дефиниране на тип данни *FileSystem*:

Може да бъде един от следните два варианта:

*File String String*: Представлява файл, който съдържа две текстови стойности:

* Първата - низ с името на файла (напр. *"readme.md"*).
* Втората - низ със съдържанието на файла (напр. *"This is a basic file system structure."*).

*Root String [FileSystem]*: Представлява директория/папка с две части:

* Низ с името на директорията (напр. *"projects"*).
* *[FileSystem]* е списък от поддиректории или файлове вътре в тази директория.

*deriving(Show)* позволява автоматично да се генерира текстово представяне на данните за печат.

2. Реализация на *stack* чрез използване на списъци:

Функциите за head, push, pop и top взаимствани от упражнения по Логическо програмиране, проведени през летен семестър 2022/2023. Преименувани са, за да препокритие с Prelude модула.

2.1. headStack: Връща елемента на върха на стека, ако има такъв. Входът е списък от произволен тип [a]. Резултатът е Maybe a, което означава: Just x, ако стекът не е празен, и Nothing, ако стекът е празен.

Пр. headStack [1, 2, 3] -- Резултат: Just 1

2.2. pushStack: Добавя нов елемент el към върха на стека. Входовете са елемент el и стек/списък st Резултатът е нов стек/списък със el, добавен накрая.

Пр. pushStack 4 [1, 2, 3] -- Резултат: [1, 2, 3, 4]

2.3. popStack: Премахва елемента на върха на стека. Входът е стек/списък st. Резултатът е нов стек без последния елемент.

Пр. popStack [1, 2, 3] -- Резултат: [1, 2]

2.4. topStack: Връща последния елемент в стека/списъка. Входът е стек/списък. Резултатът е елементът на върха.

Пр. topStack [1, 2, 3] -- Резултат: 3

3. Валидация чрез FileSystem:

3.1. isNameFolder: Проверява дали даденото име съвпада с името на директория/папка. Първият аргумент е името, което се проверява. Вторият аргумент е файловата система. Резултатът е True, ако съответното име съвпада с името на папката, или False в противен случай.

Пр. isNameFolder "projects" (Root "projects" []) -- Резултат: True

Пр. isNameFolder "documents" (File "resume.pdf" "My resume") -- Резултат: False

3.2. isNameFile: Проверява дали даденото име съвпада с името на файл. Първият аргумент е името, което се проверява. Вторият аргумент е файловата система. Резултатът е True, ако съответното име съвпада с името на файла, или False в противен случай.

Пр. isNameFile "resume.pdf" (File "resume.pdf" "My resume") -- Резултат: True

Пр. isNameFile "welcome.txt" (Root "archives" []) -- Резултат: False

3.3. isFilePath: Проверява дали даден низ представлява път. Аргументът е низ, който трябва да се провери. Резултатът е True, ако низът започва със /, или False в противен случай.

Пр. isFilePath "/home/user/documents" -- Резултат: True

3.4. isValidName: Проверява дали дадено име (на файл или папка) е валидно в контекста на текущата файловата система. Първият аргумент е функция (напр. isNameFile или isNameFolder), която проверява дали дадено име съществува. Вторият аргумент е името за проверка. Третият аргумент е файловата система. Резултатът е True, ако името не съвпада с името на текущата папка и не се среща сред елементите в нея. Ако името на текущата папка name2 е равно на name1, функцията връща False - името не е валидно, тъй като вече съществува. За останалите елементи в директорията се използва foldr: foo name1 x проверява дали даден елемент съответства на name1.

Пр. isValidName isNameFile "newfile.txt" (Root "documents" [File "resume.pdf" "My resume"])

-- Резултат: True (файлът "newfile.txt" не съществува)

4. Парсър:

Взаимствано от <https://github.com/tsoding/haskell-json/commit/bafd97d96b792edd3e170525a7944b9f01de7e34>

4.1. Дефиниция: Това е нов тип данни Parser, който е обвивка около функция с тип: String -> Maybe (String, prs). Тази функция приема входен низ от тип и връща Maybe (String, prs): Ако парсирането е успешно, връща Just (remainingString, parsedResult), където: remainingString е низът, който остава след като е извършено парсирането. parsedResult е резултатът от парсирането. Ако парсирането не успее, връща Nothing.

Линк: hackage.haskell.org -> Data.Functor

4.2. Functor Parser: Позволява прилагане на функция към резултата от парсера, като се запази структурата на парсера. fmap прилага функция foo към резултата от парсера p, където p е функция от тип String -> Maybe (String, a). След като е извлечен резултата (inputCopy, x) от парсера, прилагаме foo към x и връщаме нов парсер с резултат от тип b.

Линк: hackage.haskell.org -> Control.Applicative

Пр. uppercaseParser :: Parser String

uppercaseParser = fmap (map toUpper) wordParser

4.3. Applicative Parser: Позволява композиция на парсери. Това е важно за комбиниране на парсери, които парсират различни части от входния низ. pure създава парсер, който винаги връща дадена стойност x, без да се променя входния низ. (<\*>) позволява прилагане на парсер, който връща функция (като резултат), върху парсер, който връща аргумент. Първият парсер (left) парсира функция от тип a -> b. Вторият парсер (right) парсира аргумент от тип a. Резултатът е нов парсер, който връща резултат от тип b.

Линк: stackoverflow.com/questions/26002415/what-does-haskells-operator-do

Пр. -- Парсър за събиране на две числа.

addParser :: Parser (Int -> Int -> Int)

addParser = pure (+)

-- Използване на парсера за събиране с аргументи.

sumParser :: Parser Int

sumParser = (+) <$> addParser <\*> intParser <\*> intParser

4.4. Alternative Parser: Предоставя възможност за избор между два парсера. Ако първият парсер не успее, може да се опита вторият. empty е парсер, който винаги не успява и връща Nothing. (<|>) позволява да комбинирате два парсера така, че ако единият не успее, да се опита вторият. Ако и той не успее, резултатът ще бъде Nothing.

Пр. -- Парсър, който се опитва да парсира числа, ако не успее, преминава към буквени низове.

numberOrWordParser :: Parser String

numberOrWordParser = many digit <|> many letter

5.