

# Файлова система

---

Проста имплементация на in memory файлова система. Курсов проект по Функционално програмиране @ ФМИ

## Стартиране на проекта

За да бъде стартиран проекта под Windows, трябва да отворите терминала в директорията, в която са изтеглени трите файла: `essentials.hs`, `interface.hs`, `types.hs` и да въведете следните команди:

```
ghci -w ./interface.hs
run "/" root
```

Командата `exit` служи за прекратяване изпълнението на проекта.

## Описание и идея на проекта

Целта на проекта е да се реализира опростена имплементация на UNIX based файлова система. Проектът записва информацията за файловата система в паметта. Програмата предлага конзолен интерфейс, като поддържа следните команди:

- `pwd` (Print working directory) - изпечатва текущата работна директория
- `cd` (Change directory) - променя текущата работна директория
- `ls` (List contents) - извежда списък със съдържанието на директорията
- `cat` - конкатенира съдържанието на избрани файлове
- `rm` (Remove) - изтрива избран файл

## Проблеми при реализацията

Един от най-големите проблеми при имплементацията на проекта беше, че данните в Haskell са непроменими(immutable). Това поражда нуждата когато искаме да променим даден елемент от дървото, репрезентиращо, файловата система трябва да обходим цялата структура и да създадем копие на текущата версия, но с променен желания елемент. Друг проблем, който породил невъзможността за промяна на данни в Haskell бе, че променливата, съхраняваща текущата директория и текущата версия на файловото дърво трябва да бъдат подавани като аргумент на повечето функции, което възпрепятства на четимостта на кода.

## Структура

Проектът е разделен в три отделни модула:

- модул, отговарящ за предефинираните типове `Types.hs`
- модул, отговарящ за основните функции `Essentials.hs`
- модул, отговарящ за конзолния интерфейс `Interface.hs`

## Реализация

## Модул за типовете

При стартиране на проекта в паметта се заделя предварително създадено дърво със следната структура:

```

/
├── Folder1
│   ├── Folder1.1
│   │   └── ()
│   └── File1
│       └── (Content of file1)
├── File2
│   └── (Content of file2)
└── Folder2
    ├── File2.1
    │   └── (Content of file2.1)
    └── File2.2
        └── (Content of file2.2)

```

Предефинираните типове за подобряване четимостта на кода са следните:

### ► Предефинирани прости типове

```

type Content = String
type FileName = String
type DirName = String
type PathStr = String
type PathArr = [PathStr]

```

Простите типове са предефинирани с цел по-добра четимост. Нуждата от два типа за път - с масив и символен низ, произлиза от удобството на използването рекурсия, използвайки списъци в Haskell.

Сложният тип, дефиниран в задачата е типът, репрезентиращ системен елемент (**SystemElement**). Той може да бъде един от двата подтипа:

- Файл - Съхраняващ име(Символен низ) и съдържание(Символен низ)
- Директория - Съхраняващ име(Символен низ) и съдържание(Масив от системни елементи)

## Модул за основните функции

За всяка една от основните команди е реализирана съответна функция. Ще разгледаме някои от тях:

### 1. Функцията **changeDirectory**

```

-- !FULL PATH
getDir :: PathArr ->SystemElement ->SystemElement
getDir [] currentDir = currentDir
getDir (elder:children) currentDir = if nextDir /= dummy then getDir

```

```

children nextDir else dummy
  where nextDir = getDirByName elder currentDir

getParentPath :: PathArr -> PathArr
getParentPath ["/"] = ["/"]
getParentPath [path] = ["/"] --if only one dir -> return root
getParentPath path = init path

goToPath :: PathStr -> SystemElement -> SystemElement
goToPath "/" currentRoot = currentRoot
goToPath pathStr currentRoot = getDir pathArr currentRoot
  where pathArr = convertPathToArr pathStr

-- Current path is FULL Path
changeDirectory :: PathStr -> PathStr -> SystemElement -> SystemElement
changeDirectory pathToGo currentPath currentRoot = goToPath (getFullPath
currentPath pathToGo) currentRoot

```

Приема директорията в която трябва да отиде, настоящата директория и настоящата версия на файловото дърво. След това с помощта на няколко помощни функции връща системен елемент - директорията, до която е стигнал методът или **dummy** директория, ако подаденият път е невалиден. **dummy** директорията представлява импровизирана симулация на **Nothing** синтаксисът в езика. Използвана е, поради факта, че до момента на предаване на проекта материалът не беше покрит и не бях напълно убеден как да го използвам сигурно.

2. Функцията за **concatenateFiles** Помощния метод **addFile** добавя файл по подадени: име, съдържание, абсолютен път(под формата на масив), и настояща директория. Типът на връщане е **SystemElement** тъй, като връща копие на старото файлово дърво, но с добавен новият файл. Методът рекурсивно проемня директориите, които има нужда да бъдат променени, останалите директории се запазват. Когато директорията е невалидна се връща копие на старото дърво, а не се добавят нови папки.

***!Ако такъв файл вече съществува, той бива презаписан!***

На основния метод - **concatenateFiles** остава само да конструира файлът, който ще бъде добавен.

3. Функцията **removeFile** действа по подобен начин на функцията за добавянето на файл - рекурсивно обхожда дървото, открива файла, който трябва да бъде премахнат и го изтрива от съдържанието на съответната директория. Методът отново връща копие на текущото дърво, но без споменатия файл.

## Модул, имплементиращ интерфейса

Този модул има за цел да имплементира REPL. Реализиран е във функцията **run**. Той прочита команда и с помощта на метода **processInput** я обработва. Той от приема входните команди, настоящия път и настоящата версия на директорното дърво и връща двойка (**newPath**, **newRoot**), нужна за обновяване на информацията в цикъла. **processInput** извиква друг помощен **runCmd**, който обработва изпълнява различни функции в зависимост от подадения аргумент. Функциите са кръстени по същия начин, както функциите в **Essentials.hs** файла, но към името им е прилепена думата "Command".

## Бъдещо развитие

Проектът има множество възможности да бъде доразвит. В истинските UNIX-базирани файлови системи има множество команди, които тук не са реализирани, например:

- `cp` Копиране на файлове
- `mv` Преместване или преименуване на файлове
- `mkdir` Създаване на нова директория
- `rmdir` Изтриване на съществуваща директория
- ...

Изброените функции могат да бъдат реализирани по подобен начин с рекурсивно обхождане на директорното дърво и прилагане на промените където е нужно.

## Използвана литература

- [Материали към курса по Функционално програмиране на спец. Информатика, спец. Компютърни науки, 2 поток и избираемата дисциплина](#)
- [Learn You a Haskell for Great Good! by Miran Lipovača](#)
- [Hoogle](#)
- [Haskell wiki](#)