Tinpro01-8 Practicumopdracht 1

W. Oele

11 april 2023

Inleiding

In deze opdracht ga je een Haskell programma schrijven dat:

- gecompileerd wordt vanaf de commandline.
- wordt uitgevoerd vanaf de commandline en dus werkt zonder de Haskell interpreter.
- derhalve aan input en output doet.
- een bestand leest, comprimeert en de gecomprimeerde versie wegschrijft in een nieuw bestand.

Uitleg: Compileren en werken met I/O.

Een Haskell programma dat zonder interpreter wordt uitgevoerd zal "iets" aan I/O moeten doen daar het anders niet veel nut heeft. Dit werkt in Haskell als volgt: We schrijven een module genaamd `Main`. In deze module bevindt zich de `main` functie. Elk Haskell programma dat wordt uitgevoerd, begint simpelweg bij de `main` functie die zich in de `Main` module bevindt:

```
module Main where


main = do
  putStrLn "Hello world..."
```

Bovenstaand programma maakt gebruik van do notatie. Hoewel dat in bovenstaand stukje code eenvoudig te lezen is, zit er een hoop geraffineerde techniek achter deze notatiewijze. De precieze technische werking hiervan zal later in deze module worden uitgelegd.

De `putStrLn` functie is te vinden in de prelude van Haskell en print uiteraard een string op het scherm. Bovenstaand programma kunnen we vanaf de commandline compileren en uitvoeren op de volgende wijze:

```
wessel@digitalsnail: ghc program.hs -o program
[1 of 1] Compiling Main             ( program.hs, program.o )
Linking program ...
wessel@digitalsnail: ./program
Hello world...
wessel@digitalsnail:$
```

De optie `-o` geven we aan de compiler mee om duidelijk te maken dat we het bestand `program.hs` compileren naar het "object" `program`.

Uitleg: Bestanden lezen en schrijven.

In de prelude bevinden zich twee functies die we kunnen gebruiken voor het lezen en schrijven van bestanden:

- `readFile :: FilePath -> IO String`
- `writeFile :: FilePath -> String -> IO ()`

In bovenstaande typesignatures:

- is `FilePath` een synoniem voor `String`
- zijn de returntypes `IO String` en `IO ()`

Het voert op dit moment te ver om uit te leggen wat de returntypes van beide functies precies zijn. Ook hier geldt dat je tegen het einde van deze module precies zult weten wat het is en hoe het werkt. In deze opdracht wordt alleen uitgelegd hoe je het kunt gebruiken.

Een bestand lezen en schrijven werkt als volgt:

```
module Main where

import Data.Char

main = do
  text <- readFile "pad/naar/bestand.txt"
  let processed = reverse $map toUpper text
  writeFile "/pad/naar/anderbestand.txt" processed
  putStrLn "file written..."
```

In bovenstaande code wordt de inhoud van `bestand.txt` opgevangen in de variabele `text`. Deze variabele is van het type `String`. In de regel daarna wordt de text omgedraaid middels de `reverse` functie en worden van alle letters hoofdletters gemaakt. Het resultaat wordt in de variabele `processed` opgeslagen. De string `processed` wordt vervolgens weggeschreven in een bestand en het programma eindigt met een eenvoudige `putStrLn`.

Opdracht 1

Importeer de library `System.environment`. Je vindt in die library een functie, waarmee je argumenten die je aan je programma meegeeft, kunt opvangen in een lijst. Schrijf het programma `sortfile` dat:

- De naam van een tekstbestand als eerste argument neemt.
- het betreffende tekstbestand inleest.
- De letters uit dit bestand sorteert.
- de naam van een doelbestand als tweede argument neemt.
- De gesorteerde tekst wegeschrijft naar dit doelbestand.

Het runlength compressie algoritme: Uitleg

In deze opdracht ga je m.b.v. Haskell een klein onderzoek uitvoeren naar de effectiviteit van een bekend algoritme: Het *runlength* algoritme dat gebruikt wordt voor datacompressie. Onder *datacompressie* verstaat men het representeren van een gegeven hoeveelheid informatie in een kleiner aantal bits of symbolen. Datacompressie kent veel toepassingen, bijvoorbeeld:

- bestanden inpakken: winzip, rar, arj, gzip, etc.
- audio: mp3, flac, etc.
- foto: jpeg
- video: h264, avi, mpeg, mp4, mkv, etc.

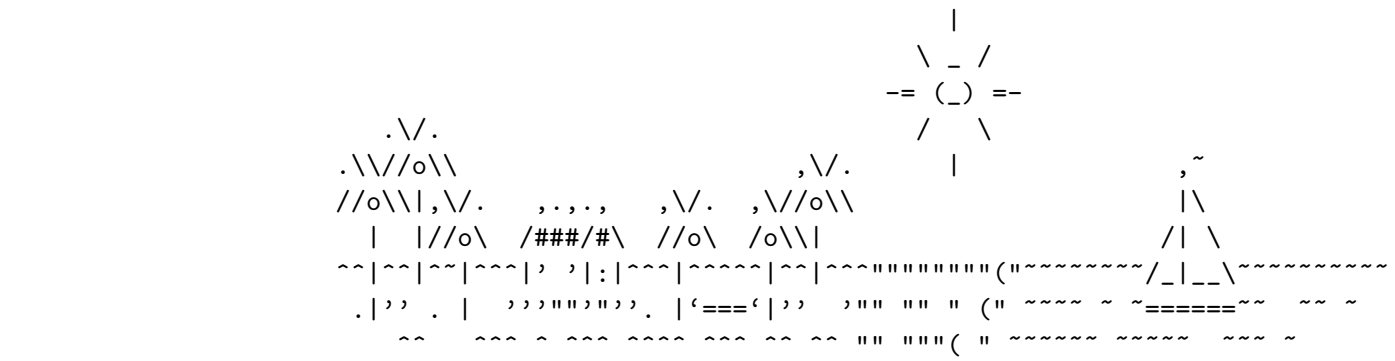
In deze opdracht houden we het eenvoudig en werken we alleen met tekstbestanden.

Het run-length algoritme: werking

Het run-length algoritme is een eenvoudig algoritme. Stel dat we in een tekstbestand het volgende tegenkomen:

```
aaabbcbbccccbbccccccaaaaabbbb
```

Een dergelijke opeenvolging van characters is niet ongewoon. Zo zitten er in onderstaand stukje ASCII art nogal wat zich herhalende characters:



Let erop dat in bovenstaand stukje ASCII art nogal wat spaties zitten en ook die zijn voor een computer "gewoon" een symbool. Andere voorbeelden zijn er natuurlijk ook: zo zitten er in diverse foto' s nogal wat opeenvolgende pixels met dezelfde kleur, etc.

Het run-length algoritme werkt als volgt: Tel het aantal zich herhalende symbolen en noteer dit met een getal, gevolgd door het betreffende symbool. De eerder genoemde string wordt zodoende als volgt gecomprimeerd:

```
aaabbcbbccccbbccccccaaaaabbbb
```

```
3a2bc2b5c2b6c5a4b
```

Het aantal symbolen in de string bedraagt 30 en het aantal symbolen na compressie is slechts 17. Van de oorspronkelijke grootte is nog maar $\frac{17}{30} \times 100 = 57\%$ over.

Het decomprimeren werkt uiteraard in de omgekeerde volgorde: 3a wordt omgezet in aaa, 5c in ccccc, etc.

Opdracht 2a: Run-Length compressie

Schrijf een programma dat een eenvoudig tekstbestand comprimeert. Voorwaarden:

- Noem het programma rlcompress
- Het programma draait stand-alone, dus zonder interpreter.
- Het programma krijgt op de commandline twee parameters mee: De naam van het te comprimeren bestand, gevolgd door de naam van het bestand, waarin de gecomprimeerde gegevens moeten worden opgeslagen.

- Het programma berekent de compressiefactor zoals boven beschreven en print deze op het scherm.
- Test je programma met een eenvoudig ASCII textbestand dat minstens 500 woorden bevat. Let erop dat in het tekstbestand geen cijfers voorkomen (waarom is dat een probleem?).
- Test je programma ook op een stuk ASCII art. Ook hier geldt dat de tekst geen cijfers mag bevatten.

Een uitvoer van het programma ziet er, bijvoorbeeld, als volgt uit:

```
wessel@digitalsnail: ./rlcompress text.txt compressed.txt
length of text.txt: 4881 characters
length of compressed file compressed.txt: 2354 characters
factor: 2354/4881*100=48%
done...
```

Opdracht 2b: Run-Length decompressie

Schrijf een programma dat een eenvoudig tekstbestand decomprimeert. Voorwaarden:

- Noem het programma rldecompress.
- Het programma draait stand-alone, dus zonder interpreter.
- Het programma krijgt op de commandline twee parameters mee: De naam van het te decomprimeren bestand, gevolgd door de naam van het bestand, waarin de gedecomprimeerde gegevens moeten worden opgeslagen.
- Test je decompressie programma met de eerder gecomprimeerde tekst en ASCII art en controleer of de gedecomprimeerde bestanden hetzelfde zijn als hun originelen.

Tips

Run-Length decompressie is lastiger te programmeren dan compressie. Lees:

- in de `Prelude` over de `show` en `read` functies.
- in `Data.List` over functies, waarmee je lijsten op allerlei manieren in stukken kunt hakken.