Berthold Hoffmann <hof>
Christian Maeder <maeder>
Klaus Lüttich <luettich>
Pascal Schmidt <pharao90>
Felix Beckwermert <foetus>
Dennis Walter <dw>
Franck Ngueuleu <nngf>

5. Übungsblatt

Ausgabe: 12.1.2004

Bearbeitungszeit: Zwei Wochen

11 *Ein/Ausgabe per* `do`*-Notation* *5 Punkte*

Schreiben Sie mit Hilfe von `putStrLn` und `getLine` ein interaktives Programm `main` in einem `Main` Modul, das kontinuierlich eine Eingabezeile (von der Konsole) erfragt und zu jeder Zeile ihre Länge und die Anzahl der Wörter (auf der Konsole) anzeigt. Die Eingabe von "quit", "q" oder "exit" beendet das Programm.

Am Ende des Programms soll ein Protokoll der Sitzung, das alle Eingaben enthält, mittels `writeFile` in eine Datei geschrieben werden. Der Dateiname wird dabei dem Programm auf der Kommandozeile übergeben. (Ohne Kommandozeilenargument wird keine Protokolldatei angelegt und weitere Kommandozeilenargumente können ignoriert werden.)

Importieren Sie `getArgs` aus dem Modul `System`, um auf die Kommandozeilenargumente zugreifen zu können.

12 *Tabelleninspektion* *10 Punkte*

Lesen Sie mit `readFile` eine Textdatei ein, deren Name auf der Kommandozeile übergeben wurde. Die Wörter jeder Zeile bilden nun die Einträge einer Tabelle gemäß Aufgabe 7.4 vom 3. Übungsblatt. Implementieren Sie dazu eine interaktive Kommandoschleife, die folgenden Aktionen ermöglicht:

- Das Programm beenden.
- Den Inhalt einer Zelle anzeigen. Dazu ist das Indexpaar zu erfragen.
- Die Spaltenbreite für nachfolgende Ausgaben erfragen.
- Mehrere Zellen einer Zeile ausgeben. Dazu ist ein Indexpaar und die Anzahl der Zellen zu erfragen.

- Einen Tabellenbereich ausgeben. Dazu sind zwei Indexpaare zu erfragen.

Für diese Aufgabe sollen genau die in Aufgabe 7.2 und 7.3 implementierten Funktionen `getRow`, `printRow` und `printTable` Verwendung finden.

Das Umwandeln eines Strings in eine (nicht-negative) Zahl (vom Typ `Int`) kann mit der Funktion `read` erfolgen, nachdem zuvor sicher gestellt wurde, dass der String nur aus 1 bis 9 Ziffern besteht. Alternativ kann folgende Implementierung von `readEither` übernommen werden:

```
readEither :: Read a => String -> Either String a
readEither s = check [ x | (x,"") <- reads s ]
    where check [x] = Right x
          check []  = Left "readEither: no parse"
          check _   = Left "readEither: ambiguous parse"
```

Hinweise: Die beiden `Main` Module der vorherigen Aufgaben werden für den `ghc` am besten in verschiedenen Verzeichnissen verwaltet, um Konflikte mit einer erzeugten Datei `Main.hi` zu vermeiden. Ansonsten muss der Dateiname für ein `Main` Module nicht `Main.hs` lauten.

Ein so genanntes "stand-alone" Programm erhält man durch die Übersetzung mit dem `ghc`:

```
ghc --make Main.hs -o main
./main <Kommandozeilenargumente für main>
```

oder durch `runhugs`:

```
runhugs Main.hs <Kommandozeilenargumente für main>
```

Wenn man eine *literate* Haskell Datei `Main.lhs` hat, kann man diese unter Unix (mit '`chmod`') ausführbar machen und als erste Zeile folgendes eintragen:

```
#!<hugspath>/runhugs
```

Den Pfad `<hugspath>` zu `runhugs` kann man per '`which runhugs`' ermitteln. Danach ist folgender Aufruf möglich:

```
./Main.lhs <Kommandozeilenargumente für main>
```

Dieses ist Version 2.0 von 12. Januar 2004.