5 *par2sfs, awk* und *sed*

Loggen Sie sich ein und gehen Sie ihr zugewiesenes Gruppen-Directory (z.B. cip3). Arbeiten Sie *immer* nur in Ihrem Gruppen-Directory.

```
cip1 % cd SS08/cip3
```

5.1 Visualisierung einer Partitur: par2sfs und Es

In Ihrem Directory befinden sich folgende neue Dateien:

`g121axx0_000_OLV.nis` :	Sounddatei im NIST Format
`g121axx0_000_OLV.par` :	Zugehöriges Partitur-File

Schauen Sie sich das Partitur-File an und identifizieren Sie die eingetragenen Spuren.

```
cip1 % less g121axx0_000_OLV.par
```

Hören Sie sich das Signal an.

```
cip1 % test_nist p=1 g121axx0_000_OLV.nis
```

Wandeln sie das Partitur-File und das zugehörige Signalfile in ein SFS-Archiv-File um.

```
cip1 % par2sfs
cip1 % par2sfs sptype=nist g121axx0_000_OLV.par g121axx0_000_OLV.nis
...
```

Der Befehl par2sfs zeigt Ihnen, welche tiers er erfolgreich einbinden konnte. Schauen Sie sich das SFS-Archiv-File mit dem Display-Programm 'Es' an:

```
cip1 % Es -t g121axx0_000_OLV.sfs
```

Spielen Sie ein wenig mit dem Beipiel herum: probieren Sie die Cursor aus (Linke und rechte Maustasten), Zoomen Sie hinein (Z), wieder heraus (T), hören Sie sich einzelne Segmente an, etc.

5.2 Arbeiten mit 'gawk'

'Gawk' ist ein Programm, das genau wie 'grep' Files zeilenweise nach Mustern (regulären Ausdrücken) absucht und für jedes Muster eine bestimmte Aktion durchführen kann. Der simpelste Aufruf von 'gawk' ist

```
cip1 % gawk <Programm> <Input-File>
```

Ein GAWK-Programm besteht immer aus folgenden Kommandos:

```
<Muster1> { Aktion1 }
<Muster2> { Aktion2 }
...
```

'gawk' sucht dann in jeder Zeile des Files nach den Mustern und wenn es eines davon findet, wird die entsprechende Aktion dahinter ausgeführt.

Zum Beispiel gibt folgender Befehl

```
cip1 % gawk '/^ORT/ { print $3 }' g121axx0_000_OLV.par
```

nur die dritte Spalte (`$3`) aus allen Zeilen, die mit 'ORT' beginnen auf den Bildschirm (standard output).

Wenn man das Muster weglässt, wird jede Zeile verarbeitet:

```
cip1 % gawk '{ print $3 }' g121axx0_000_OLV.par
```

Bei längeren Programmen ist es handlicher, diese nicht immer wieder einzutippen, sondern in einem File zu speichern. Außerdem will man fast immer viele Files auf einmal in einer Pipe verarbeiten. Der Aufruf sieht dann so aus:

```
cip1 % cat file1 file2 file3 ... | gawk -f <Programm-File>
```

Schreiben Sie mit einem Editor (z.B. pico) folgendes Programm in das File hesi.awk (achten Sie darauf in Ihrem Gruppen-Dir zu sein!):

```
BEGIN { count = 0 }
/^ORT:.*<"ah>/ {
          count = count + 1
        }
/^ORT:.*<"ahm>/ {
          count = count + 1
        }
/^ORT:.*<hm>/ {
          count = count + 1
        }
END   { print "Anzahl der Hesitationen: " count }
```

Rufen Sie es auf mit dem Befehl

```
cip1 % cat ../Partitur/* | gawk -f hesi.awk
```

Genau wie andere Skriptsprachen kennt gawk Variablen und Kontroll-Kommandos (if-Verzweigung, Schleifen, etc.). Eine vollständige Beschreibung der Sprache ist in der Manual-Page von gawk zu finden:

```
cip1 % man gawk
```

Erweitern Sie Ihr Programm-File so, daß es außerdem noch die Anzahl der Wörter insgesamt ausgibt (zweiter Zähler, Muster für ORT-Zeilen).

Erweitern Sie Ihr Programm-File so, daß es außerdem noch die relative Anzahl von Hesitationen pro Wort ausgibt.

Testen Sie nun alle Sprecher (w.o.), Münchner Sprecher (`cat ../Partitur/m???d*`), Karlsruher Sprecher (`cat ../Partitur/n???k*`), Bonner Sprecher (`cat ../Partitur/m???n*`) und Kieler Sprecher (`cat ../Partitur/j???a*`)

Wer verwendet im Mittel mehr Hesitationen?

5.3 Automatisches Editieren mit 'sed'

Wir haben schon in den ersten Stunden sog. Filter kennengelernt, die Operationen auf Text-Files automatisch ausführen können. Bestes Beispiel für ein Filterprogramm ist der UNIX-Befehl 'tr' (translate), der einzelne Zeichen vertauschen, zusammenfassen ('squeezen') oder löschen kann.

```
cip1 % echo "Mein Geld" | tr 'M' 'D'
```

Manchmal reicht die einfache Funktionalität von 'tr' nicht mehr aus, z.B. wenn man längere Zeichenketten ersetzen möchte. Nehmen wir an, wir haben einen Fehler in unseren Partitur-Files entdeckt. Und zwar hat jemand konsequent `'n"ahmlich'` statt `'n"amlich'` geschrieben. Wir können das z.B. für die Münchner Files feststellen mit dem Befehl:

```
cip1 % grep 'n"ahmlich' ../Partitur/m???d*
```

Wir können jetzt nicht einfach mit 'tr' alle 'h'-Zeichen löschen; also brauchen wir einen komplexeren Mechanismus, der sämtliche Editierfunktionen beherrscht. Der *stream editor* 'sed' ist so ein komplexes Filter. Genauso wie 'tr' ließt er von standard input, filtert das File und schreibt es wieder nach standard output.

```
cip1 % echo 'n"ahmlich'
cip1 % echo 'n"ahmlich' | sed 's/n"ahmlich/n"amlich/'
```

Der allgemeine Aufruf von 'sed' ist sehr ähnlich wie bei 'gawk':

```
cip1 % cat file | sed <Programm>
```

Das 'Programm' bei 'sed' besteht nur aus einzelnen Editierbefehlen. Von allen Befehlen der wichtigste (und am häufigsten gebrauchte) ist der 'substitute' Befehl:

```
s/muster/replacement/
```

Also wird in obigem Beispiel der String `'n"ahmlich'` durch den String `'n"amlich'` ersetzt. Ein kleines angehängtes 'g' ('global') bewirkt, dass alle vorkommenden Muster in einer Zeile ersetzt werden (sonst nur das erste auftretende Muster):

```
s/muster/replacement/g
```

Der Muster-Teil des 'substitute' Befehls ist wie bei 'gawk' oder 'grep' ein regulärer Ausdruck. Nochmal zur Wiederholung:

`^` :	Zeilenanfang
`$` :	Zeilenende
`.` :	ein beliebiges Zeichen
`*` :	eine beliebige Wiederholung
`[xyz]` :	Eines von 'x' oder 'y' oder 'z'
`[a-g]` :	Ein Zeichen von 'a' bis 'g'
`[^xyz]` :	Ein Zeichen, das nicht 'x' oder 'y' oder 'z' ist
`\( \)` :	Musterklammerung (kann im Replacement-Teil mit `'\1'` wieder eingesetzt werden)

Kopieren Sie sich das fehlerhafte File in Ihr Gruppen-Dir. Ersetzen Sie mit 'sed' in dem fehlerhaften File alle falschen `'nähmlich'` durch `'nämlich'`.

6 Übung zu *sed* und *awk*

6.1 Aufgabe

Stellen Sie sich vor, die Steuerfahndung steht vor der Tür und sie haben auf Ihrem PC noch sämtliche gefälschten Bilanzen gespeichert. Einfach Löschen geht nicht, weil die Beamten dann misstrauisch werden. Sie müssen ganz schnell handeln, weil Sie die Steuerfahnder schon auf der Treppe hören.

1. Entwerfen Sie ein sed Programm, das im eingegebenen Text sämtliche vorkommende EUR-Beträge bis 999 EUR durch 'XXX' ersetzt, also z.B.

   ```
   EUR 445,78   ->   EUR XXX,78
   EUR 87,00    ->   EUR XXX,00   (beachten Sie, daß das Komma noch an der richtigen Stelle steh
   ```

   Andere Zahlenangaben sollen aber erhalten bleiben, z.B.

   ```
   '4. Vorstandssitzung'  ->  '4. Vorstandssitzung'
   ```

   Testen Sie Ihren Befehl mit der Datei 'Bilanzen' in Ihren Gruppendirectory.

   ```
   % cat Bilanzen | sed 'PROGRAM' | less
   ```

2. Sind Beträge mit verschieden langen Lücken zwischen 'EUR' und dem Betrag korrekt abgebildet worden? Z.B.

   ```
   'EUR       45,78'   ->   'EUR XXX,78'
   ```

   Wenn nicht, erweitern Sie den Befehl dazu, daß diese auch erfasst werden. Könnten Sie Ihren Befehl so modifizieren, daß die Länge der Lücke genau gleich bleibt?

3. Jetzt sollen auch Tausender und Millionenbeträge (auf die kommt es ja an!) alle korrekt abgebildet werden, z.B.

   ```
   'EUR  1.900.000,-  ->  EUR XXX,-
   ```

4. Das folgende macht zwar keinen unmittelbaren Sinn, wenn die Steuerfahndung kommt, aber es ist eine schöne Übung: Schreiben Sie ein gawk Programm, das im Inputfile nach allen EUR-Beträgen sucht und diese korrekt zusammenaddiert und schließlich die Summe ausgibt.

   Tip: Schreiben Sie Ihr gawk Programm nicht mehr auf die Kommandozeile (wie in den Beispielen oben), sondern in ein Programm-File. Der Aufruf ändert sich dann zu:

   ```
   % cat Bilanzen | gawk -f PROGRAMM-FILE
   ```

   Auf diese Weise können sie leichter mehrzeilige Programme schreiben und müssen nicht dauernd auf der Kommandozeile editieren.

   Gehen Sie schrittweise vor:

   - Schreiben Sie zunächst ein Programm, das alle Zeilen mit EUR Beträgen ausgibt (Denken auch mal an grep und eine Pipe-Struktur)

- Erweitern Sie es, damit nur die EUR Beträge ausgegeben werden (die einzelnen FELDER einer Zeile werden in awk durch die Variablen $1, $2, ... repräsentiert)
- Schauen sie sich in der Man Page zu gawk (`% man gawk`) die String-Funktionen gsub() an und überlegen, wie Sie die EUR-Beträge in ganze Zahlen umformatieren können. Geben Sie sie zur Kontrolle wieder aus.
- Schließlich addieren Sie sie auf und geben das Endergebnis aus.

5. Irgendein Computer-Oldtimer überedet Sie, Ihre Bilanz in LaTeX umzuformatieren. Unter anderem müssen dazu alle Umlaute anders geschrieben werden:

   ```
   ä -> "a
   Ü -> "u
   ö -> "o
   ß -> "s
   Ä -> "A
   Ü -> "U
   Ö -> "O
   ```

   Z.B.

   ```
   'Ötzi ißt Übel.'  ->  '"Otzi i"st "übel'
   ```

   Schreiben Sie ein gawk Programm, das diese Übersetzung automatisch macht und testen Sie es mit Ihrer Bilanz.

   Tips:

   - `$0` ist die Variable, die die gesamte Input-Zeile enthält
   - Ein doppeltes Hochkomma (") muß in einem String immer mit \ quotiert werden, z.B. `"R\"ugen"`, damit es nicht mit dem Ende des Strings verwechselt wird.

6. Schreiben Sie ein gawk Programm, daß im Input-Text sämtliche Zahlen (nur Kardinalzahlen) in ausgeschriebene Zahlwörter umwandelt. Also z.B.:

   ```
   'Ich esse 46 Bananen.'  ->  'Ich esse sechsundvierzig Bananen.'
   ```

6.2 Lösungen

1. ```
   cat Bilanzen | sed 's/EUR [0-9]*,/EUR XXX,/' | less
   ```

2. ```
   cat Bilanzen | sed 's/EUR *[0-9]*,/EUR XXX,/' | less
   cat Bilanzen | sed 's/\(EUR *\)[0-9]*,/\1XXX,/' | less
   ```

3. ```
   cat Bilanzen | sed 's/\(EUR *\)[0-9.]*,/\1EUR XXX,/' | less
   ```

4. —

5. Siehe File `~/WS06_07/cip1/latex.awk`

6. Siehe File `/share/local/lib/digit2words.awk`
   ```