Beispiel

Beispiel: Namensliste konvertieren (Karl Egon Meier $\rightsquigarrow$ Meier, Karl Egon).

```
s/(.*) (.*)/$2, $1/;
```

Problem: mehrteilige Nachnamen (von Goethe, Mac Donald, Di Caprio)

```
s/(.*) (.*)/$2, $1/;
s/(.*) ([a-z]+|Ma?c|D[aei]|L[ae])$/$2 $1/;
```

Tip: Daten vorher „desinfizieren“:

```
s/\t/ /g; ## Tab -> Leerzeichen
          ##   (g = global, d.h.: alle Vorkommen ersetzen)
s/ +/ /g; ## beliebige Folge von Leerzeichen
          ##   -> ein Leerzeichen
s/^ //;   ## Leerzeichen am Anfang löschen
s/ $//;   ## Leerzeichen am Ende löschen
```


Beispiel

Beispiel: Erstes und zweites Argument einer Methode f(x,y,z) vertauschen.

Problem: Um Aufrufe wie `f(g(x,h(y)),z,w)` richtig umformen zu können, müßte man Klammern zählen. Das geht mit regulären Ausdrücken nicht.

Wir können aber immerhin vorher überprüfen, ob solche kritischen Fälle in der Datei auftauchen:

Taucht eine öffnende Klammer vor dem ersten Komma nach „`f(`“ auf?

```
\bf\s*\([^(,]*\(
```

Taucht eine öffnende Klammer zwischen dem ersten und dem zweiten Komma auf?

```
\bf\s*\([^,]*,[^(,]*\(
```

Wenn nein, dann funktioniert folgende Ersetzung:

```
s/\bf\s*\(([^),]*),([^),]*)/f($2,$1/g;
```


Reguläre Ausdrücke in der Praxis

Schlußbemerkungen:

- Es ist unnötig, die Fähigkeiten und syntaktischen Details *aller* existierenden Regexp-Implementierungen zu lernen. Eine oder zwei (ein Editor + eine Programmiersprache) sind allerdings nützlich. (Üben!)
- Ergebnisse immer kontrollieren.
- Die meisten Regexp-Implementierungen benutzen NEAs. Die Suche mit komplizierten regulären Ausdrücke kann darum sehr viel Zeit benötigen (insbesondere: Vereinigung und Abschluß geschachtelt). Oft besser: Problem aufteilen.
- Bei Verwendung regulärer Ausdrücke in Programmen: unbedingt kommentieren!

Programmiersprachen: C und C++

Historische Entwicklung

C (Kernighan und Ritchie, ∼ 1974)

imperative Programmiersprache, maschinennah,
einfach zu compilieren.

C++ (Stroustrup, ∼ 1983)

objektorientierte *Erweiterung* von C: „C mit Klassen“.

Java (Gosling et al., ∼ 1993)

objektorientierte Programmiersprache mit an C und C++ angelehnter Syntax, aber ohne volle Kompatibilität mit C.

Was aus C ist in Java nicht enthalten?

Zeigervariable:

In Java:

primitive Datentypen: Variable enthält Wert.
sonstige Datentypen: Variable enthält Zeiger auf Wert.
(andere Möglichkeiten existieren nicht)

In C (und C++):

```
int n;      // n enthält ganze Zahl.
int* pn;    // pn enthält Zeiger auf ganze Zahl.
int** ppn;  // ppn enthält Zeiger auf Zeiger auf
            // ganze Zahl.
```

(gilt so für *alle* Datentypen).

Was aus C ist in Java nicht enthalten?

Referenzierungs- und Dereferenzierungsoperatoren:

`*px`   liefert das, worauf die Zeigervariable `px` zeigt.

`&x`   liefert die Adresse der Variablen `x`.

Was aus C ist in Java nicht enthalten?

```
void swap(int* px, int* py) {
    int hilf;
    hilf = *px;
    *px = *py;
    *py = hilf;
}

void main() {
    int a = 1;
    int b = 2;
    printf("a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
```

Was aus C ist in Java nicht enthalten?

Zeigerarithmetik:

In C verhalten sich Arraynamen und Zeiger beinahe gleich:

```
int a[10];        /* a ist ein Array, nicht wie in Java
                     eine Referenz auf ein Array! */
int* pn;
int n;
int i;

n = a[0];         /* a[0] ist eine int-Variable; n bekommt
                     den Wert dieser Variable zugewiesen. */

pn = &a[0];       /* pn = Adresse der Variablen a[0]. */
n = *pn;          /* n = Wert, auf den pn zeigt. */

pn = a;           /* a und &a[0] sind gleichwertig! */
n = *pn;
```


Was aus C ist in Java nicht enthalten?

```
n = a[i];

pn = &a[i];       /* pn = Adresse der Variablen a[i]. */
n = *pn;          /* n = Wert, auf den pn zeigt. */

pn = &a[0];       /* pn = Adresse der Variablen a[0]. */
n = *(pn + i);    /* pn + i ist ein Zeiger auf die Variable,
                     die i Plätze hinter pn liegt. */

pn = a;           /* wie oben: a und &a[0] sind gleichwertig! */
n = *(pn + i);

n = *(a + i);     /* der Name eines int-Arrays kann wie ein
                     Zeiger auf ein int verwendet werden. */

pn = &a[0];       /* und umgekehrt: die Variable pn
n = pn[i];           (Zeiger auf int) kann wie ein
                     int-Array verwendet werden. */
```


Was aus C ist in Java nicht enthalten?

Der C-Compiler erlaubt potentiell unsichere Typ-Umwandlungen:

```
char* calloc();
int* pn;

pn = (int*) calloc(n, sizeof(int));
     /* calloc liefert einen Zeiger auf genügend
        Speicherplatz für n Werte der angegebenen
        Größe. */
```

Es ist beispielsweise auch möglich, Zeiger in Zahlen zu konvertieren (oder umgekehrt).

Was aus C ist in Java nicht enthalten?

In C gibt es keinen expliziten `boolean`-Typ.
Zahlen werden bei Bedarf als logische Werte interpretiert:
0 ist falsch, andere Zahlen sind wahr.

(In neueren C++-Versionen gibt es den Typ `bool` mit den Elementen `true` und `false`.)

Was aus C ist in Java nicht enthalten?

Eine Abkürzung:

Wenn `ps` ein Zeiger auf eine `struct` (oder in C++ ein Objekt) ist, dann müßte man auf die Komponente `xy` dieser `struct` oder dieses Objekts mittels `(*ps).xy` zugreifen.

Da diese Situation extrem häufig vorkommt, gibt es dafür eine Abkürzung: `ps->xy`

Was aus C ist in Java nicht enthalten?

Der C-Präprozessor:

Bevor der eigentliche C-Compiler ein Programm übersetzt, wird die Datei erst durch einen Präprozessor geschickt, der einige Ersetzungen im Quelltext vornimmt.
Präprozessor-Anweisungen beginnen mit „#“:

```
#define NMAX 1024
/* im Rest der Datei wird NMAX durch 1024 ersetzt. */

#include "dateiname"
/* die angegebene Datei wird an dieser Stelle eingefügt. */
```


Was aus C ist in Java nicht enthalten?

```
#ifdef BEZEICHNER
...
#else
...
#endif
/* falls BEZEICHNER dem Präprozessor bekannt ist, werden
   die Zeilen zwischen #else und #endif ignoriert,
   anderenfalls die Zeilen zwischen #ifdef und #else.
   Verschachtelte #ifdef sind möglich.
   Übliche Anwendung: Unterschiede zwischen verschiedenen
   Prozessoren oder Betriebssystemen. */
```