Skriptsprachenorientierte Programmiertechnik

Perl Teil 6

Prof. Ilse Hartmann

Gliederung

Reguläre Ausdrücke

- **Idee:** Beliebige Textkette kann mittels beliebigem Suchmuster durchsucht werden.

- **Entweder passt das Suchmuster, oder nicht.**

- **Länge und Art der Muster frei.**

- **Hinweis: Globbing != Mustersuche**
  (Globbing = Mustererkennung für Dateinamen, z.B. *.doc)

Einführung

- ***regular expression***, häufig ***RegExp*** oder ***RegEx*** abgekürzt:
  - + **Muster oder „Schablone**“ zur Untersuchung von Zeichenketten
  - + Ein String **„passt“** zum Muster oder passt nicht
  - + Reguläre Ausdrücke stehen in der Regel zwischen Slashes: ***/<regexp>/***
  - + Auch nach **Escape-Zeichen (z.B. \t oder \n**) kann gesucht werden
  - + Pattern Matching ist standardmäßig **case-sensitiv!**

- Neben diesem ***pattern matching*** existieren weitere vielfältige Möglichkeiten, z.B. **Suchen&Ersetzen (*substitution)***

- Im Prinzip ähnlich der Nutzung von Wildcards (*, ?) beim DIR-Befehl von DOS

- Vgl. auch das Unix-Kommando grep

RegEx	Untersuchter String	Match?
/ist/	Perl **ist** großartig!	**TRUE**
/Perl/	**Perl ist großartig!**	**TRUE**
/perl/	**Perl ist großartig!**	**FALSE**
/\n/	**Dieser String besteht aus...¶** ... zwei Zeilen!	**TRUE**
/r\tS/	**Ein kleiner<*tab*>Schritt...**	**TRUE**

Pattern Matching

- Zum Vergleich mit **$_** wird das Muster zwischen **Slashes** geschrieben

- Der **Bindungsoperator =~** vergleicht Text auf der linken Seite mit dem Muster rechts

- Der **Operator !~** negiert den Vergleich

- Mit Option **/i** wird case-insensitiv gesucht (groß/klein ignoriert)

Beispiele:

```
$_ = "Perl ist großartig!";
if (/ist/) {
        print "Das Muster wurde gefunden!\n";
}
my $satz = "Ein Satz mit \tTab und \nZeilenumbruch.";
if ($satz =~ /\tTab und \n/) {
        print "Das Muster wurde gefunden!\n";
}
if ($satz !~ /gibt's nicht/) {
        print "Kein Match für das Muster.\n";
}
if ($satz =~ /ein/i) {
        print "Das Muster wurde gefunden!\n";
}
```

Metazeichen

- **Diverse Zeichen haben in regulären Ausdrücken eine spezielle Bedeutung**

  + Der **Punkt (.)** steht z.B. für genau(!) ein(!) beliebiges(!) Zeichen (außer *Newline)*

  + Zur Suche nach literalen Zeichen, die eine Sonderfunktion ausüben, wird **der Backslash (\)** verwendet

  + Zur Suche nach dem literalen Backslash muß dieser seinerseits durch einen **Backslash** geschützt werden

Metazeichen

Irgendein Zeichen

Der Punkt

RegEx	Untersuchte Strings	Match?
/i.t/	Perl ist großartig! Egon teilt aus. Fritz muß warten. Hefe quillt auf. I.g.i.t.t!	**TRUE** **TRUE** **FALSE** **FALSE** **TRUE**
/i\.t/	Perl ist großartig! I.g.i.t.t! I..g..i..t..t!	**FALSE** **TRUE** **FALSE**
/\\/	Hier kommt ein Backslash(\) Dies ist nur ein Slash: /	**TRUE** **FALSE**

Quantifier

- Zur Suche nach einer bestimmten Anzahl von Zeichen werden **Quantifier** eingesetzt

- **Quantifier** bezeichnen generell die Anzahl des voranstehenden Zeichens (bzw. Ausdrucks)

Quantifier	Bedeutung
+	Mindestens einmal, beliebig oft
*****	Beliebig oft (auch nur einmal oder überhaupt nicht!)
?	Genau einmal oder gar nicht

Quantifier

RegEx	Untersuchte Strings	Match?
/an+e/	Fritz ißt eine Banane. Die Wanne ist voll. Michaela singt.	**TRUE** **TRUE** **FALSE**
/an*e/	Fritz ißt eine Banane. Die Wanne ist voll. Michaela singt.	**TRUE** **TRUE** **TRUE**
/an?e/	Fritz ißt eine Banane. Die Wanne ist voll. Michaela singt.	**TRUE** **FALSE** **TRUE**

Zeichenklassen

- **Liste von Zeichen**, die alternativ für **genau ein Zeichen** stehen

- **Zeichenklassen** stehen in **eckigen Klammern ([ ])**

- **Zeichenbereiche** können mit **Bindestrich (-)** angegeben werden

RegEx	Bedeutung
[abcwxyz]	Findet eines dieser 7 Zeichen
[a-cw-z]	Abkürzung des ersten Beispiels
[a-zA-Z]	Findet alle Buchstaben (=52 Zeichen)
[0-9]	Findet alle Ziffern
[A-Za-z0-9]	Findet alle alphanumerischen Zeichen (Zahlen/Buchstaben)

Zeichenklassen

Für einige gängige Zeichenklassen existieren Abkürzungen

Abkürzung	Steht für	Bedeutung
\d	**[0-9]**	Beliebige Ziffer (*digit)*
\w	**[A-Za-z0-9_]**	„Wort“-Zeichen, alle alphanumerischen Zeichen und der Unterstrich
\s	**[\f \t \n \r]**	Alle *Whitespace-Zeichen: Seitenvorschub, Tabulator,* Zeilenumbruch, Wagenrücklauf, Leerzeichen

Negieren von Zeichenklassen

- Findet alle Zeichen außer den angegebenen

- Durch Voranstellen des **Caret-Zeichens (^)** innerhalb der eckigen Klammern

- Negierte Abkürzungen: Durch Großbuchstaben **(\D, \W und \S)**

RegEx	Bedeutung
[^abc]	Findet alle Zeichen außer a, b und c
[^a\-z]	Findet alle Zeichen außer a, Bindestrich (-) und z
[^0-9]	Findet alle Nicht-Ziffern
[^\d]	Findet alle Nicht-Ziffern
\D	Findet alle Nicht-Ziffern
\W	Findet alle Zeichen, die nicht alphanumerisch sind (Ausnahme: Unterstrich)
\S	Findet alle Zeichen, die **kein Whitespace sind**

Anker

- Beschränken Treffer auf den Anfang und/oder das Ende eines Strings
- Zur Suche am Anfang einer Zeichenkette: **Caret-Zeichen (^)**
- Bei Angabe des Dollar-Zeichens **($)** werden Treffer nur am String-Ende berücksichtigt
- Ein etwaiges **Newline-Zeichen** am String-Ende wird hierbei ignoriert

RegEx	Untersuchte Strings	Match?
/^Perl!/	Perl! Es ist großartig! Es lebe Perl!	**TRUE** **FALSE**
/Perl!$/	Perl! Es ist großartig! Es lebe Perl! Es lebe Perl! ¶	**FALSE** **TRUE** **TRUE**
/^Perl!$/	Perl! Es lebe Perl! Perl!	**FALSE** **TRUE**

Einfache Mustererkennung

Beispiel vgl. mit $_:

if (*/pattern/*){ print “Gefunden!“; }

wobei *pattern irgendein Text sein kann, piffi, 1\n2 usw.*

- **Metazeichen, z.B.:**
  - + **.** Ein Zeichen, außer \n, so genannte Wildcard
  - + **** nimmt Metazeichen die Sonderbedeutung
  - + **()** Zeichenkette als ein Zeichen auffassen
- **Quantifizierer (Quantifier)**
  - + ***** 0-n Vorkommen des vorangehenden Zeichens, z.B. \t*
  - + **+** 1-n Vorkommen des vorangehenden Zeichens
  - + **?** 0-1 Vorkommen des vorangehenden Zeichens
  - + Hinweis: Alle Quantifizierer dürfen nicht(!) am Anfang es Musters stehen!
- **Frage: Was findet /(piffi)*/ ?**

Einfache Mustererkennung

- **Zeichenklasse: Liste von Zeichen in [ ], die mit logischem Oderverknüpft sind, Beispiel:**
  - + **[abdNM]** a, b, d, N oder M
  - **+ [a-gN-Z]** a, b, c, .., g, N, M, ...Y, Z
  - + Wichtig **- als [\-]**
- **Negation mit Caret ^, Beispiel:**
  - + **[^abc],** alles außer a, b, oder c
- **Abkürzungen:**
  - + **[0-9]** == \d # Ziffernklasse
  - **+ [A-Za-z0-9_]** == \w # Wortklasse,
  - + **[\f\t\n\r]** == \s # Whitespaceklasse
  - + Beispiel: Anreden aus Briefen finden: **/Hallo \w+,/**
  - + **[^\d] == [\D], [^\w] == [\W], [^\s] == [\S]**

Split

- **Trennt einen String** in eine Liste von Einzelstrings auf

  Beispiel:

  @array = split *lsomeSplittingSign/, $string;*

- Das **Trennzeichen** wird durch einen **RegEx** beschrieben

- Wird **kein String** angegeben, wird **$_** gesplittet

- **split ohne Argumente** trennt **$_ an Whitespaces**
  (jedoch ohne führende/folgende Leer-Elemente)
- **Trennung an 1-n Whitespaces:**

  split /\s+/, $*someString;* **bzw:**

  my @array = split; # entspricht /\s+/, $_;

- **Resultat liegt in den Elementen eines Arrays.**

Split

Beispiel:

```
@woerter = split(/:/, "abc:def:g:h");
# ergibt (“abc“, “def“, “g“, “h“)

@woerter = split(/:/, "abc::def:g:h");
# ergibt (“abc“, ““, “def“, “g“, “h“)

@woerter = split(/\s+/, “\tHallo, dies\tist\nein Test!\n");
# ergibt (““, “Hallo,“, “dies“, “ist“, “ein“, “Test!“)

$_ = "\tHallo, dies\tist\nein Test!\n";
@woerter = split;
# ergibt (““,“Hallo,“, “dies“, “ist“, “ein“, “Test!“)
```

Join

- **Fügt Listen wieder zu einem String zusammen**

- **Das erste Argument ist das Trennzeichen (normaler String, hier kein RegEx)**

- **Gegenteil zu split:**

```
my $result = join $glue, @pieces;
my $a = join ” ”, ”Hallo“, ”Kurs“, ”!“; # ”Hallo Kurs !“
count pieces-1 == count glue
```

Join

Beispiele:

```
my @woerter = split(/:/, "abc:def:g:h");
print join("-", @woerter);
# „abc-def-g-h“
print join("-", "a", "b");
# „a-b“
print join("-", "a");
# „a“
my @leer; # leeres Array
print join("-", @leer);
# leerer String
```

Mustersuche mit //

- **m// Mustervergleichsoperator, auch // oder m{}, oder m!! usw.**
- **Vorsicht bei Suche nach einem Zeichen, dass als Trennzeichenbenutzt wird.**
- **Beispiel: /^http:\/\// == m%^http://% Hinweis: / ist kein(!) Metazeichen!**

- **Option-Modifier / Flags (stehen hinter dem schließenden Trennzeichen und verändern das Default Verhalten):**
  + **/i** Keine Unterscheidung der Groß- und Kleinschreibung
  + **/s** Beliebiges Zeichen bei Punktsymbol, Punktsymbol passt dann auch auf Zeilenende
  + **/x** Leerzeichen im Muster, z.B.:

```
/-?\d+\.?\d*/ == / -? \d+ \.? \d* /x, auch:
/
-?          # optionales Minus
\d+         # 1-n Ziffern vor dem Dezimalpunkt
\.?         # optionaler Dezimalpunkt
\d*         # Ende des Musters
/x          # Ende
```

Hinweis: # könnte auch als Trennzeichen verwendet, oder im Muster gesucht werden!

+ Mehrere einfach hintereinander schreiben.

Mustersuche

- **Verankern:**
  + Muster an bestimmter Stelle im Suchstring finden:
    - **• ^ Beginn des Strings, also *Fred Geröllheimer wird gefunden, nicht aber Manfred***
    - ***Mann mit /^fred/i***
    - **• $ Ende des Strings**

  + Wörter an Wortgrenzen finden mit \b, also Anfang und/oder Ende eines Wortes aus [\w], also /\w+/. Jeder String hat eine gerade Anzahl an Wortgrenzen, da jedes Wort Anfang und Ende haben muß.

    Beispiele:
    - **• /\b*some_word/ Am Anfang***
    - **• /*some_word\b/ Am Ende***
    - **• /\b*some_word\b/ Genau das Wort***

- **Bindungsoperator =~:**
  Die Mustererkennung, die sonst mit $_ durchgeführt wird, soll stattdessen mit dem String links von =~ durchgeführt werden.

Beispiel:

+ if($*some_string* =~ */\bABC/)*

Finde ABC am Wortanfang in $some_string

+ **my $var = <STDIN> =~ /\bYES\b/i;**
  # Wahrheitswert wird abgelegt, wenn Yes (u.a.) eingegeben wird.

- **Variableninterpolation in Suchmustern,**

Beispiel grep:

```
my $was = “piffi“;
while(<>){
        if(/^($was)/){
                # Suchmuster am Anfang des Strings verankern
                print “Am Anfang von $_ steht $was“;
        } }
```

- **Hinweis:** Im Suchmuster keine Zeichen eingeben die alsTeil des Musters erkannt würden, wie z.B. (
- **Speichervariablen:** Enthalten den gefundenen String (sind also skalare Variablen), heißen $1, $2 usw.

  je nachdem Speicherklammern (und) im Suchmuster existieren.

Beispiel:

```
$_ = “Hallo Piffi, alles klar“;
if(/(\S+) (\S+) (\S+) (\S+)/){
        print “Ich habe $1 $2 $3 $4 gefunden“; }
```

Mustersuche

```
my $dinoJung = “Ich sterbe in 1000 Jahren“;
    my $dinoAlt = “Ich sterbe in einer Million Jahre“;
    If($dinoJung =~ /(\d*) Jahren/{
        print “Noch $1 Jahre übrig“; # 1000
    }
    If($dinoAlt =~ /(\d*) Jahren/{
        print “Noch $1 Jahre übrig“; # undef, also Empty String
    }
```

- **Speicher bleibt bis zum nächsten erfolgreichen(!) Mustervergleich erhalten:**

```
    if($somethg =~ /(\w+)/){
        print “$1“;}
    else {
        print “Nothing in it“;}
```

Automatische Speichervariable

- **$&** Derjenige Teil auf den ein Suchmuster tatsächlich zugetroffen hat.

- **$`** Alles was vor dem Gefundenen im String stand. Bzw. alles was übersprungen werden musste bis das Suchmuster gefunden wurde,

- **$‘** Alles was nach dem Gefundenen im String stand. Bzw. alles was nie erreicht wurde.

Allgemeine Quantifier

- Bestimmte Anzahl innerhalb eines Suchmusters, bisher **?*+**

- Alternativ: ***{Mindestens, Höchstens},***
  Beispiel: *la{2,5}l* - zwischen 2 und 5 Vorkommen von a

- Auch ***{Mindestens,}, keine Obergrenze***

- Auch ***{Genau}*** *Beachte: Kein Komma in der Klammer*

- Also:
  - *** entspricht {0,}**
  - **+ entspricht {1,}**
  - **? Entspricht {0,1}**

Präzedenz in Suchmustern

1. **()** Gruppierung von Ausdrücken
2. **Quantifier,** hängen immer mit dem Element zusammen auf das sie folgen
3. **Anker und Zeichenfolgen:**

   - Anker

   - Caret ^ (Stringanfang)
   - Dollarzeichen $ (Stringende)
   - Anker \b (Wortgrenzen) und \B (Nicht Wortgrenzen)

   - Zeichenfolgen: Die Buchstaben eines Wortes halten genauso eng zusammen, wie die Anker mit einem Wort zusammenhängen.

Präzedenz in Suchmustern

4. Alternativen |,

wg. /abc|def/ wäre dann keine Auswahl wie abc oder def, sondern /ab c oder d ef/ statt /abc oder def/

- **Beispiele:**

  + **/^fred|barney$/**

  findet: fred *irgendwas anderes* barney\n

  + **/(wilma|pebbles?)/**

  Findet: wilma, pebbles oder pebble,

  da ? Eine höherer Präzedenz hat als |

- **Bisher nur Suche. Jetzt Bearbeitung beim Gefundenen.**

- Ersetzungen mit **s///: m//** wie Suche,
  s/// wie Suche und Ersetze das erste Vorkommen.
  Beispiel:

```
$_ = “Hallo Piffi“;
s/Piffi/Puffi/; # Piffi durch Puffi ersetzen, also
print “$_\n“;
```

- Wenn keine Ersetzung möglich ist, bleibt die Variable unverändert.
  Liefert in jedem Fall einen Boole‘schen Wert, ob Ersetzung geklappt hat, oder nicht.

Globales Ersetzen mit /g

- **/g Modifier,** Beispiel**: s/Hans/Peter/g;**
- Viele Whitespace auf 1 „ „ reduzieren: **s/\s+/ /g;**
- Führende Whitespace Zeichen entfernen: **s/^\s+//;**
- Nachgestellte Whitespace Zeichen entfernen: **s/\s+$//;**
- **Beides: s/^\s+|\s+$//g;**
- **Andere Begrenzungszeichen:**
  + Nicht paarweise Zeichen: **s#^https://#http://#;**
  + Paarweise Zeichen, paarweise anwenden: **s[^https://]{http};**

- Optionsmodifier:

  + **/i** Keine Unterscheidung der Groß- und Kleinschreibung
  + **/s** Beliebiges Zeichen
  + **/x** Leerzeichen im Muster

- **Bindungsoperator: =~ wie bei s///,**
  z.B.: $fileName =~ s#^.*/##s; # rm Unixpath from name

Groß- und Kleinschreibung

- **\U (Wort) \u (1. Buchstabe) UPPERCASE**
- **\L \l lowercase**
- **Per Default Wirkung auf den Rest des Strings.**
- **\E nur das erste Vorkommen (Barney im obigen Beispiel)**
- **Funktionieren bei allen Strings mit “ “,**

  Beispiel:

  print “Hallo, \L\u$name\E, wie spät ist es?“;

m // im Listenkontext

- **Rückgabewert im Listenkontext ist eine Liste der bei der Musterkennung angelegten Speichervariablen (oder eine leere Liste).**
- **Beispiel:**

```
$_ = “Hallo, werter Nachbar!“;
my ($a, $b, $c) = /(\S+), (\S+) (\S+)/;
print “$a, mein $c\n“;
```

Nicht-gierige Quantifier

- **Gieriger Quantifier:** Es wird der längst mögliche String gefunden.

- **Nicht-gieriger Quantifier:** Zeichen eines Teiltreffers wieder „freigeben“.

- **Beispiel: “Fred und Barney waren gestern beim Bowling“**

  Suchmuster: /Fred.+Barney/

  Vorgehen:

  + Fred wird gesucht und gefunden.
  + Dann .+ (1-n Zeichen (ohne \n)) da + gierig sind das ALLE!
  + Dann Barney: Wegen nicht-gierigem Verhalten wird iterativ von hinten gesucht ob Barney gefunden werden kann. (Backtracking)

Nicht-gierige Quantifier

Für jeden gierigen Quantifier gibt es auch ein nicht gieriges Gegenstück, hier: .+?

+ Vorgehen hier ab „ und“ usw.
+ Auch hier Backtracking, aber wegen Position schneller gefunden.

- Weitere Nicht-gierige Quantifier:

+ ***?***
+ ***(sth)?***
+ ***??*** Passt einmal oder gar nicht, wobei gar nicht bevorzugt wird