Manfred Rosendahl

Pattern Matching

Zusammenfassung
Es werden Verfahren zur Texterkennung und --Bearbeitung mit Hilfe von an die Programmiersprache SNOBOL angelehnte Klassen zum Pattern-Matching beschrieben. Die Klassen sind gegenüber dem SNOBOL Vorbild zum Teil abgeändert, um eine größere Effizienz zu erzielen.

(1) Pattern-Matching

Beim Pattern-Matching in Texten beschäftigt man sich mit der Definition von Testmustern (Pattern), die durch Kombination in der Lage sind, frei vorkommende Teststrukturen zu beschreiben. Die Muster und ihre Kombinationen sollen durch geeignete Algorithmen erkannt werden können. Solche Patterns sind Bestandteile vieler Programmier-- und Skript-Sprachen. Die Sprache JAVA kennt z.B. die Klasse **tokenizer**, wo ein Text anhand von Trennzeichen in Teile zerlegt werden kann. Die Skriptsprache PERL hat mit ihren regulären Ausdrücken (regex: Perl regular expression) ebenfalls ein recht mächtiges Hilfsmittel Pattern zu beschreiben und zu erkennen. Hiermit können jedoch nur reguläre Strukturen erkannt werden.
Eine der ersten Sprachen, gerade auf diese Art der Textverarbeitung ausgerichtet, war SNOBOL. Der im Folgenden beschriebene Ansatz benutzt im Wesentlichen die in SNOBOL definierten Patterns. Er definiert jedoch nicht eine neue Programmier - oder Skriptsprache, sondern die Implementation erfolgt über in einer vorhandenen Sprache (PASCAL-DELPHI) geschriebene Klassen und Funktionen. Dadurch kann die Methodik in jedem Programm, das in diesen Sprachen geschrieben ist, genutzt werden. Eine Implementierung z.B. in Java oder C++ wäre ebenfalls möglich.

Patterns können u.a. sein:

- Strings einer vorgegebenen Länge
- alle Zeichenfolgen bis zu einem speziellen Zeichen
- die längste Folge von Blanks
- beliebige Wiederholungen eines Strings
- Strings balanciert bezüglich von vorgegeben Klammersymbolen
- beliebige Strings

Aus den Basis-Patterns können durch Alternativen (ALT) und Folgen (CAT) komplexere Pattern gebildet werden.
Beispiel:

```
p:=alt(['be','bea','bear']);
q:=alt(['ro','roo','roos']);
r:=alt(['ds','d']);
s:=alt(['ts','t']);
pa:=alt([pat([p,r]),pat([q,s])]);
```

`pa` match jeden der folgenden Strings:

```
beds        rots
bed         rot
beads       roots
bead        root
beards      roosts
beard       roost
```

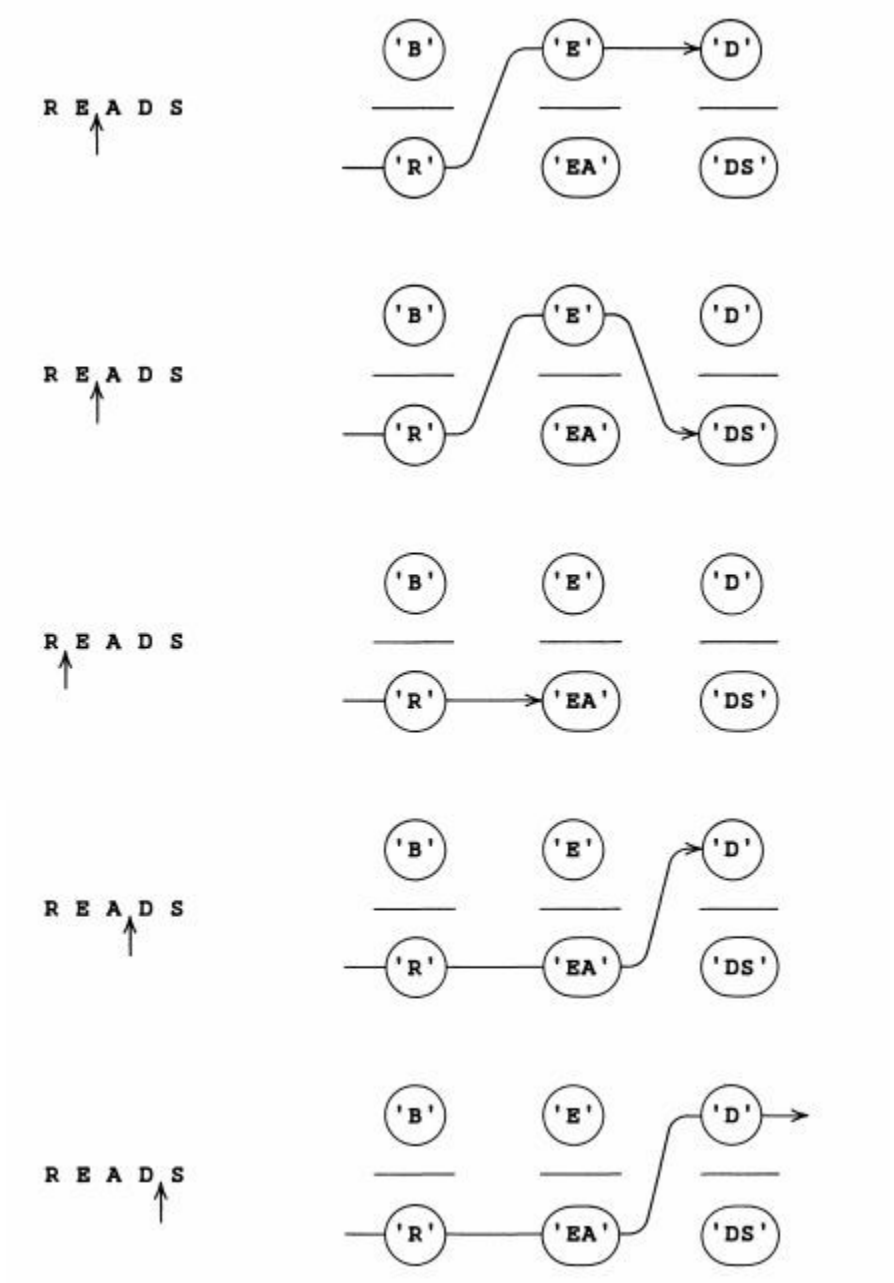
Eine Prozedur zur Durchführung des Pattern-Matching ist:
function mat(var s:string;ab:integer;const pat:tpattern;var apos,alen:integer):boolean;

overload;
Es wird im String s ab der Position ab versucht das Pattern pat zu matchen. Ist dies erfolgreich wird die Position und die Länge des gematchten String auf apos und alen zurückgegeben. Bei Matching gibt es zwei Varianten:
Nach anchor(1) muss der Pattern im String exakt bereits ab der Position gematcht werden. Nach anchor(0) kann der Pattern irgendwo im String s ab der Position gematcht werden.

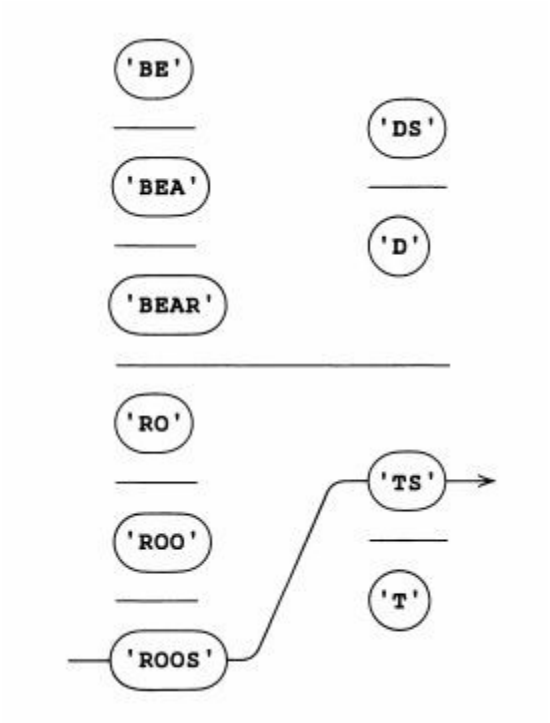
Das Pattern Matching kann man sich vorstellen, dass eine Nadel durch die passenden Patterns geführt wird. Kommt man an einer Stelle nicht mehr weiter, wird sie zurückgezogen und es wird bei dem vorangegangenen Pattern die nächste Alternative genommen. Existiert dort keine mehr, wird bei dem davor liegenden Pattern die nächste Alternative genommen. Entweder kann das gesamte Pattern gematcht werden oder beim ersten Pattern gibt es keine Alternative mehr.
Dies soll an folgendem Beispiel erläutert werden:
Das Pattern

```
br:=pat([alt(['b','r']),alt(['e','ea']),alt(['d','ds'])]);
```

soll getestet werden gegen den String: `'reads'`



Testet man das obige Pattern `pa` gegen den String `roosts` so benötigt man 11 Schritte.



Der Aufwand kann also im ungünstigsten Fall exponentiell mit der Länge des String steigen. In vielen fällen kann man jedoch Pattern nutzen, die keine Alternativen haben, z.B. Suchen eines Pattern. Ferner wird für alle Pattern die Mindestlänge der matchenden String berechnet. Dadurch können Alternativen von vornherein ausgeschlossen werden. Auch wird versucht möglichst mit Klassen auszukommen, die keine oder nur wenige Alternativen haben. Am ungünstigsten ist das Pattern arb, das jeden String erkennt. Mit Hilfe des Patterns find (siehe weiter unten) kann hierauf meist verzichtet werden.

Implementiert wird der Suchalgorithmus, indem für jede Patternklasse eine Methode matchfirst existiert.

function matchfirst(astr:string;apos,arlen:integer;var alt,len:tobject) :Boolean;

Die Funktion versucht den String astr ab der Position apos zu matchen. arlen gibt an, welche Länge die nachfolgenden Patterns mindestens haben. Auf den Parametern alt und len wird behalten welche Alternative erkannt wurde und wie lang der gematchte Teilstring ist. Dies wird benötigt zum Zurücksetzen und um die nächste Alternative auszuwählen.

Das Erkennen der nächsten Alternative erfolgt mit der Methode:

function matchnext(astr:string;apos,arlen:integer;var alt,len:tobject) :Boolean;

Entsprechend dem Parameter alt wird versucht die nächste Alternative zu erkennen. Gibt es bei dem Pattern keine Alternative, so liefert matchnext immer false.

(2) Beschreibung der Grundpattern

Alle Pattern bilden abgeleitete Klassen, der abstrakten Grundklasse `tpattern` . Die Grundpatterns können zu einer Folge zusammengefasst werden. Beim Matching der Folge müssen die einzelnen Patterns hintereinander gematcht werden. Ferner können mehrere Patterns alternativ verknüpft werden und ein Pattern kann negiert werden.

Bei der Definition eines Patterns kann eine Stringvariable angegeben werden. Auf diese Variable wird der gematchte Teilstring zugewiesen.
Für alle Patternklassen existieren Konstruktoren und für diese wird jeweils erzeugende Funktionen definiert.

STRI ,SV,CAT,SF

function stri(const astr:string):tstri;

Bei Pattern tstri wird der Wert des String jedoch bei der Erzeugung des Pattern festgelegt. Möchte man den Inhalt des zu matchenden String erst während der Matching Prozedur festlegen, muss man eine Stringvariable benutzen.

function sv(var av:string):tsv;overload;

In der Funktion matchfirst wird der Wert der Variablen mit dem zu testenden String verglichen. Dazu wird jeweils, die Methode compute benutzt.
Die Klasse wird z.B. benutzt, wenn ein Anfangstag erkannt wurde und dann der zugehörige Endtag gesucht wird.
Von der Klasse tsv können durch Überschreiben der Methode auch eigene Klassen abgeleitet werden, die aus beliebigen Argumenten den zu testenden String berechnen.
Eine bereits definierte abgeleitete Klasse ist tcat.
Möchte man den String nicht nur mit einer Stringvariablen vergleichen, sondern mit der Verkettung mehrerer benutzt man:

function cat(aarr:array of const):tcat;overload;

Der Parameter von cat ist eine Liste von String bzw. Objekten vom Typ tsv-
Beispiel:
cat([sv(s1), 'xyz', sv(s2)])
Das Ergebnis von compute angewandt auf dieses Objekt ist die Verkettung s1+'xyz'+s2. Der Wert der Variablen wird jeweils zum Zeitpunkt des Aufrufs der Matching-Methode genommen.

Es gibt auch eine Klasse tsf, die eine vom Benutzer zu definierende Funktion zur Berechnung des Teststring benutzt.

type tstringfunction=function(s:string):string;

function sf(af:tstringfunction;var s:string):tsf;

Die bei compute benutzte Funktion f kann eine beliebige Funktion sein, die aus einem Argumentstring den Teststring berechnet. Sie könnte z.B. alle Kleinbuchstaben in Großbuchstaben wandeln.

IV,MINUS,SUMME

Möchte man testen, ob eine erst zur Laufzeit festgelegte Zahl im Teststring vorkommt, kann man die Klasse tiv benutzen.

function iv(var av:integer):tiv;overload;

Es gibt zwei von tiv abgeleitete Klassen, die eine Summe bzw. einen negativen Wert berechnen, der dann im Teststring vorkommen muss.

function summe(aarr:array of const):tsumme;overload;

function minus(av:tiv):tminus;

BAL,SBAL

Um Klammerstrukturen zu erkennen, benötigt man Pattern, die balancierte String erkennen. Balancierte Strings werden für Klammersymbole definiert. Ein String ist balanciert, wenn am Ende die Anzahl der öffnenden und schließenden Klammern gleich ist. Ferner darf dazwischen die Zahl der schließenden Klammern niemals größer als die Zahl der öffnenden Klammern sein. Dies kann auch definiert werden für mehrere Klammerpaare.
z.B. seinen die Klammern ( ) und [ ].
Dann wäre der String: **`(a+(b*c[x[2]+y]))`** ist balanciert.
Der String **(...[...)...]** ist jedoch nicht balanciert.
Balancierte Strings haben also eine Wohldefinierte Klammerstruktur.

function bal(const klastr:string):tbal;overload;
function bal(const klastr:string;var s:string):tbal;overload;

function quote(const quotestr:string):tbal;overload;
function quote(const quotestr:string;var s:string):tbal;overload;

function balquote(const klastr,quotestr:string;aslash:char):tbal;overload;
function balquote(const klastr,quotestr:string;aslash:char;var s:string):tbal;overload;

function balquote(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tbal; overload;
function balquote(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char;var s:string):tbal; overload;

Mit bal(',[,]') werden String erkannt, die mit einer öffnenden Klammer beginnen und mit der zugehörenden schließenden Klammer enden. Zusätzlich muss der String eine balancierte Zahl von öffnenden und schließenden Klammern [und] enthalten.
Bei bal gibt es keine Alternativen.
Das erste Zeichen des Arguments ist der Trenner zwischen öffnenden und schließenden Klammertext, d.h. er können auch Klammern aus mehr als einem Zeichen benutzt werden. Ferner können auch mehrere Klammertexte angegeben werden. Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.
Verwandt ist ein Quotestring. Während ein Klammerstring beendet ist, wenn die Anzahl der öffnenden Klammern gleich der Anzahl der schließenden Klammern ist, ist ein Quotestring, wenn auf den öffnenden Quotetext der erste schließende Quotetext folgt, unabhängig, ob dazwischen weitere öffnende Quotetexte vorkommen. Daher kann auch der schießende Quotetext identisch mit öffnenden sein.
Bei bal und quote können auch mehrere Texte für Öffnen und Schließen benutzt werden.
Beispiele:
quote(',{,},(*,*)') erkennt die Pascal Kommentare.
bal(',(,),[,]') erkennt Klammerstrukturen mit eckigen und runden Klammern.

Bei der Funktion balquote werden natürlich innerhalb eines gequoteten Bereichs auch keine Klammersymbole erkannt.

Beim Aufruf bal(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tbal;
können auch Pattern als Klammer- bzw. Quoteelemente angegeben werden. Damit können Texte mit Anfang- und Endtags erkannt werden. So kann bei XML zu einem Anfangstag der zugehörige Endtag erkannt werden. Siehe Beispiele weiter unten.

Tritt ein Zeichen hinter einem speziellen Fluchtsymbol (z.B. Slash) auf, so soll es nicht als Quote- oder Klammersymbol erkannt werden. Dies kann durch einen zusätzlichen slash Parameter definiert werden.

bal hat hier eine etwas andere Funktion als bei SNOBOL. Dort wird jeder beliebige String erkannt, wenn innerhalb des String die öffnenden und schließenden Klammern balanciert sind. Es wird also auch ein beliebiger String erkannt, der überhaupt keine Klammersymbole enthält. Bei bal muss der zu matchende String mit einem Klammer-auf-String beginnen, gematcht wird bis zum zugehörenden Klammer-zu-String. Daher hat bal auch keine Alternativen.

Ein der SNOBOL Funktion bal äquivalentes Patern ist:
function sbal(const klastr:string):tsbal;overload;
function sbal(const klastr:string;var s:string):tsbal;overload;

function squote(const quotestr:string):tsbal;overload;
function squote(const quotestr:string;var s:string):tsbal;overload;

function sbalquote(const klastr,quotestr:string;aslash:char):tsbal;overload;
function sbalquote(const klastr,quotestr:string;aslash:char;var s:string):tsbal;overload;

function sbalquote(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tsbal;overload;
function sbalquote(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char;var s:string):tsbal;overload;

sbal entspricht dem arb (beliebiger String), nur mit dem Unterschied, dass innerhalb des gematchten String die Klammern und Quote ausgeglichen sein müssen.
Beispiel:
Bei dem String 123(')'45)678
würde das Pattern
`sbalquote(',(,)',','',''',s2)`
nacheinander folgende Teilstring matchen:

```
1
12
123
123(')'45)
123(')'45)6
123(')'45)67
123(')'45)678
```

ANY,NOTANY

Mit Hilfe der Klasse **tany** kann jedes Zeichen aus einer Zeichenmenge gematcht werden.
Die Zeichenmenge kann durch einen String oder einen set of char beschrieben werden.
Der Parameter von tany kann auch zur Laufzeit erst festgelegt werden, durch eine Instanz von tsv als Parameter. Hier wird der Wert des String zum Zeitpunkt des Tests genommen. So kann getestet werden, ob ein vorher vorkommendes Zeichen wieder vorkommt. Bei tany gibt es keine Alternative.

function any(const astr:string):tany;overload;
function any(const achars:tcharset):tany;overload;
function any(const astr:string;var s:string):tany;overload;
function any(const achars:tcharset;var s:string):tany;overload;
function any(avv:tsv):tany;overload;
function any(avv:tsv;var s:string):tany;overload;

Beispiele:
any('AEIOU') oder any(['A','E','I','O','U']) matcht jeden Vokal
any('0123456789') oder any(['0'..'9']) matcht jede Ziffer

Der Gegensatz von tany ist **tnotany**. Hier wird jedes Zeichen gematcht, das nicht aus einer vorgegebenen Zeichenmenge stammt.

function notany(const astr:string):tnotany;overload;
function notany(const achars:tcharset):tnotany;overload;
function notany(const astr:string;var s:string):tnotany;overload;
function notany(const achars:tcharset;var s:string):tnotany;overload;
function notany(avv:tsv):tnotany;overload;
function notany(avv:tsv;var s:string):tnotany;overload;

SPAN,UPTO

Mit der Klasse **tspan** werden String gematcht, die nur aus Zeichen aus einer vorgegebenen Zeichenmenge stammen.

function span(const astr:string):tspan;overload;
function span(const achars:tcharset):tspan;overload;
function span(const astr:string;var s:string):tspan;overload;
function span(const achars:tcharset;var s:string):tspan;overload;
function span(avv:tsv):tspan;overload;
function span(avv:tsv;var s:string):tspan;overload;

function anyspan(const astr:string):tspan;overload;
function anyspan(const achars:tcharset):tspan;overload;
function anyspan(const astr:string;var s:string):tspan;overload;
function anyspan(const achars:tcharset;var s:string):tspan;overload;
function anyspan(avv:tsv):tspan;overload;
function anyspan(avv:tsv;var s:string):tspan;overload;

Ein Aufruf von span matcht falls kein Zeichen aus der Zeichenmenge folgt den leeren String. Dagegen erfordert anyspan zumindest ein Zeichen aus der Zeichenmenge, sonst liefert anyspan fail als Ergebnis. Bei span(const astr:string) wird die Zeichenmenge aus den Zeichen des String gebildet. Mit span(avv:tsv) kann die dieser String, analog wie bei tany auch beim Matching gebildet werden.
tspan arbeitet immer greedy, d.h. es wird solange gematcht, wie Zeichen aus der Zeichenmenge folgen. Daher gibt es auch keine Alternativen.
Beispiele:
span(' ') Folge von Blanks
span(['0'..'9']) oder span('0123456789') Folge von Ziffern
span(['A'..'Z']) Folge von Großbuchstaben

Analog wie bei notany gibt es auch zu tspan die duale Klasse **tupto.**

function upto(const astr:string):tupto;overload;

function upto(const achars:tcharset):tupto;overload;
function upto(const astr:string;var s:string):tupto;overload;
function upto(const achars:tcharset;var s:string):tupto;overload;
function upto(avv:tsv):tupto;overload;
function upto(avv:tsv;var s:string):tupto;overload;

upto ist immer erfolgreich, da auch der leere String gematcht wird.
upto arbeitet immer greedy, d.h. es wird solange gematcht, wie Zeichen nicht aus der Zeichenmenge folgen. Daher gibt es auch keine Alternativen.
Die entsprechende Funktion heißt bei SNPBOL break. Da break aber ein Schlüsselwort in Delphi ist wurde upto genommen.

Beispiele:
upto(' ') bis zum nächsten Blank
span(['0'..'9']) oder span('0123456789') bis zum nächsten nicht Ziffer Zeichen
span(['A'..'Z']) bis zum ersten nicht Großbuchstaben

LEN

Möchte man einen beliebigen String fester Länge matchen kann man die Klasse **tlen** benutzen.

function len(ai:integer):tlen;overload;
function len(ai:integer;var s:string):tlen;overload;

Das Matching ist nicht erfolgreich, wenn der Reststring kürzer als die verlangte Länge ist.
Bei tlen gibt es keine Alternative.

Beispiel:
pat(['(',len(5),')']) matcht jeden String, wo exakt 5 Zeichen von Klammern eingeschlossen sind.

pat([len(2,t),'.',len(2,m),'.',len(4,j)])
matcht genau ein Datum im Format tt.mm.jjjj, wobei auf den Stringvariablen t,m,j die Werte abgelegt werden.

ARB

Bei der Klasse tarb wird jeder String, ab einer vorgegeben Länge gematcht. Bei arb(0) jeder String.

function arb(ai:integer):tarb;overload;
function arb:tarb;overload; // i=0 Matcht jeden String
function arb(ai:integer;var s:string):tarb;overload;
function arb(var s:string):tarb;overload; // matcht jeden String

Beim ersten Matching wird ein String mit minimaler Länge gematcht. Bei arb(0) der leere String. Bei jedem Rematch wird dann die Länge jeweils um eins erhöht. Das Matching ist nicht erfolgreich, wenn die Restlänge nicht mehr ausreicht. **arb** ist nicht sehr effizient, da von der Länge 0 an alle Längen getetet werden. Effizienter ist die Funktion f**ind** (s.u:)

TAB,RTAB,REM

Möchte man den Teilstring von der aktuellen Position bis zu einer gegebenen Position im String matchen kann man die Klassen **ttab** bzw **trtab** benutzen. Bei ttab wird die Position vom Anfang gerechnet, bei trtab vom Ende. Mit rtab(0) wird so der gesamte Reststring gematcht. Ist die

aktuelle Cursorposition schon über die angegebene Position hinaus, ist das Matching nicht erfolgreich. Statt rtab(0) kann auch rem angegeben werden.
Bei ttab bzw trtab gibt es keine Alternative.

function tab(atabi:integer):ttab;overload;
function tab(atabi:integer;var s:string):ttab;overload;
function tab(av:tiv):ttab;overload;
function tab(av:tiv;var s:string):ttab;overload;

function rtab(atabi:integer):trtab;overload;
function rtab(atabi:integer;var s:string):trtab;overload;
function rtab(av:tiv):ttab;overload;
function rtab(av:tiv;var s:string):ttab;overload;
function rem:trtab;overload;
function rem(var s:string):trtab;overload;

Bei tab(av:tiv) kann die Position auch beim Matching erst festgelegt werden, z.B. eine feste Distanz zu einer vorher mit tcurs (s.u.) ermittelten Cursorposition.
Durch einen zusätzlichen Parameter kann man, wie bei den meisten Pattern, auch angeben, auf welcher Stringvariablen der gematchte Teilstring abgespeichert werden soll.

CURS

Möchte man die aktuelle Cursorposition abspeichern kann man **tcurs** benutzen.

function curs(var apos:integer):tcurs;

curs(i) matcht immer erfolgreich den leeren String und liefert die aktuelle Cursorposition auf der Variablen i.
Beispiel:

pat([pos(0),span(letters,s),arb,sv(s),rpos(0)])

wird erfolgreich gematcht, wenn am Anfang und am Ende die gleiche Buchstabenfolge vorkommt.
Dies ist allerdings nicht sehr effizient, da bei arb viele Alternativen probiert werden.
Effizienter allerdings nicht so elegant ist:
pat([lpos(0),span(letters,s),curs(i),rtab(iv(i)),sv(s),rpos(0)])
Hierbei gibt es keine Alternativen, sodass der Aufwand liniear ist.

Dies sieht man an den Trace Ausgaben:
Bei:
pat([pos(0),span(letters,s),arb,sv(s),rpos(0)])

tlpos :[]abc12345abc
tpat first(0):[]abc12345abc
tspan first:[abc]12345abc
tpat first(1):[abc]12345abc
tarb first:abc[]12345abc
tpat first(2):abc[]12345abc
tsv first:abc[fail]12345abc
tarb next:abc[1]2345abc
tpat first back(2):abc[1]2345abc
tsv first:abc1[fail]2345abc
tarb next:abc[12]345abc
tpat first back(2):abc[12]345abc

```
tsv first:abc12[fail]345abc
tarb next:abc[123]45abc
tpat first back(2):abc[123]45abc
tsv first:abc123[fail]45abc
tarb next:abc[1234]5abc
tpat first back(2):abc[1234]5abc
tsv first:abc1234[fail]5abc
tarb next:abc[12345]abc
tpat first back(2):abc[12345]abc
tsv first:abc12345[abc]
tpat first(3):abc12345[abc]
trpos :abc12345abc[]
tpat first(4):abc12345abc[]
tpat first total:[abc12345abc]
TRUE
```

Bei:
pat([pos(0),span(letters,s),arb,sv(s),rpos(0)])

```
tlpos :[]abc12345abc
tpat first(0):[]abc12345abc
tspan first:[abc]12345abc
tpat first(1):[abc]12345abc
tcurs :abc[]12345abc
tpat first(2):abc[]12345abc
trtab :abc[12345]abc
tpat first(3):abc[12345]abc
tsv first:abc12345[abc]
tpat first(4):abc12345[abc]
trpos :abc12345abc[]
tpat first(5):abc12345abc[]
tpat first total:[abc12345abc]
TRUE
```

FPOS,RPOS

Bei einem Aufruf von fpos(i) wird ebenfalls der leere String gematcht, jedoch nur, wenn die aktuelle Cursorposition i ist.

function fpos(aposi:integer):tfpos;overload;
function fpos(av:tiv):tfpos;overload;

Bei rpos(i) wird die Cursorposition entsprechend vom Ende gerechnet.

function rpos(aposi:integer):trpos;overload;
function rpos(av:tiv):trpos;overload;

(3) Zusammengesetzte Pattern

Sequentielle Folge: PAT

Beim Patternmatching möchte man i.A. nicht nur ein Pattern erkennen, sondern bestimmte Strukturen, die durch Folgen von Pattern und Alternativen zusammengesetzt werden. Sequentielle Folgen von Pattern kann man beschreiben mit der Klasse **tpat.**

Die Liste der Pattern wird von links nach rechts zunächst mit matchfirst gematcht. Ist bei einem Teilpattern das Matching nicht erfolgreich, wird versucht für das vorangegangene Pattern mit matchnext eine Alternative zu finden. Ist matchnext erfolgreich wird wieder nach rechts weitergemacht. Ist matchnext nicht erfolgreich, weil es keine Alternative mehr gibt, wird weiter links eine Alternative gesucht. Der Prozess endet entweder damit, das alle Teilpattern gemacht wurden, dann ist das Gesamt Matching erfolgreich, oder es wurde für das erste Teilpattern keine Alternative mehr gefunden, dann ist das Gesamt Matching nicht erfolgreich. War das Gesamt Matching erfolgreich, müssen alle Alternativen, die zu dem erfolgreichen Matching geführt haben, gemerkt werden. Die Pattern Folge kann nämlich Teilpattern einer umgebenden Sequenz sein. Damit kann für die Pattern Folge auch später matchnext aufgerufen werden. Dabei muss dann, jeweils vom letzten Teilpattern beginnend, eine neue Alternative gesucht werden.

function pat(apat:array of const):tpat;overload;
function pat(apat:array of const;var s:string):tpat;overload;

Die Elemente der Liste apat können sein:
- Pattern
- Zahl i, i=0 entspricht arb(0); i>0 entspricht len(i)
- String s: entspricht stri(s), d.h. der String s wird gematcht

Die Elemente des array of const werden in eckige Klammern geschrieben.

Beispiel:
Wenn bu die Menge der Buchstaben ist und zi die Menge der Ziffern dann matcht

pat([wordstart,any(bu),span(bu+zi)])

alle Bezeichner

function pat(astr:string):tstri;overload;
matcht den String astr

function pat(ai:integer):tarb;overload;
function pat(ai:integer;var s:string):tarb;overload;
matcht jeden String mit einer Länge >=ai

function pat(apat:tpattern;var s:string):tpat;overload;
matcht das Pattern apat und speichert den gematchten String in s

function pat(apat:array of const):tpat;overload;
function pat(apat:array of const;var s:string):tpat;overload;
Matcht die Patternfolge apat

ALT Alternativen

Mit der Klasse **talt** kann man Pattern mit Alternativen beschreiben.

function alt(aalt:array of const):talt;overload;
function alt(aalt:array of const;var s:string):talt;overload;

Bei matchfirst wird zunächst die erste Alternative des ersten Pattern gematcht. Sind beim ersten Pattern alle Alternativen erschöpft, wird die erste Alternative des zweiten Pattern gematcht. Das Matching ist nicht erfolgreich, wenn alle Alternativen des letzten Pattern ausgeschöpft sind.

REP,STAR,PLUS,OPT Wiederholungen

Um reguläre Strukturen zu erkennen, muss man auch Wiederholungen von Pattern mit vorgegebener Anzahl erkennen. Dazu dient die Klasse **trep.**

function rep(apat:tpattern;avon,abis:integer;agreedy:boolean):trep;overload;
function rep(apat:array of const;avon,abis:integer;agreedy:boolean):trep;overload;
function rep(const s:string;avon,abis:integer;agreedy:boolean):trep;overload;
function rep(apat:tpattern;avon,abis:integer;agreedy:boolean;var s:string):trep; overload;
function rep(apat:array of const;avon,abis:integer;agreedy:boolean;var s:string):trep; overload;
function rep(const ss:string;avon,abis:integer;agreedy:boolean;var s:string):trep; overload;

Das Pattern pat muss mindestens von mal vorkommen und höchsten bis mal. Bei bis=0 gibt es keine obere Schranke. Ist greedy=false wird bei matchfirst die erste Alternative der minimalen Anzahl von Wiederholungen gematcht. Bei matchnext werden dann in diesen Wiederholungen von hinten her alle Alternativen gefunden. Gibt es bei allen bisherigen Wiederholungen keine Alternative mehr, dann wird die Anzahl der Wiederholungen erhöht. Bei greedy=true wird hingegen bei matchfirst versucht soviel Wiederholungen wie möglich zu matchen. Entsprechend liefert matchnext dann auch keine Alternative.

Der erste Parameter von rep kann sein
- Pattern
- Folge von Pattern, wie bei tpat
- ein String, der dann zu matchen ist

Für die gängigen Konstrukte bei regulären Ausdrücken gibt es spezielle Aufrufe:

function star(apa:tpattern):trep;overload;
function star(apa:array of const):trep;overload;
function star(const s:string):trep;overload;
function star(apa:tpattern;var s:string):trep;overload;
function star(apa:array of const;var s:string):trep;overload;
function star(const ss:string;var s:string):trep;overload;

function star(apa:tpattern;agreedy:boolean):trep;overload;
function star(apa:array of const;agreedy:boolean):trep;overload;
function star(const s:string;agreedy:boolean):trep;overload;
function star(apa:tpattern;agreedy:boolean;var s:string):trep;overload;
function star(apa:array of const;agreedy:boolean;var s:string):trep;overload;
function star(const ss:string;agreedy:boolean;var s:string):trep;overload;

star entspricht von=0 und bis=0, d.h. 0 bis beliebig viele Wiederholungen.
Wird agreedy nicht angegeben wird der Standardwert greedystd genommen.

function plus(apa:tpattern):trep;overload;
function plus(apa:array of const):trep;overload;
function plus(const s:string):trep;overload;
function plus(apa:tpattern;var s:string):trep;overload;
function plus(apa:array of const;var s:string):trep;overload;
function plus(const ss:string;var s:string):trep;overload;

function plus(apa:tpattern;agreedy:boolean):trep;overload;
function plus(apa:array of const;agreedy:boolean):trep;overload;
function plus(const s:string;agreedy:boolean):trep;overload;

function plus(apa:tpattern;agreedy:boolean;var s:string):trep;overload;
function plus(apa:array of const;agreedy:boolean;var s:string):trep;overload;
function plus(const ss:string;agreedy:boolean;var s:string):trep;overload;

plus entspricht von=1 und bis=0 d.h. 1 bis beliebig viele Wiederholungen

function opt(apa:tpattern):trep;overload;
function opt(apa:array of const):trep;overload;
function opt(const s:string):trep;overload;
function opt(apa:tpattern;var s:string):trep;overload;
function opt(apa:array of const;var s:string):trep;overload;
function opt(const ss:string;var s:string):trep;overload;

function opt(apa:tpattern;agreedy:boolean):trep;overload;
function opt(apa:array of const;agreedy:boolean):trep;overload;
function opt(const s:string;agreedy:boolean):trep;overload;
function opt(apa:tpattern;agreedy:boolean;var s:string):trep;overload;
function opt(apa:array of const;agreedy:boolean;var s:string):trep;overload;
function opt(const ss:string;agreedy:boolean;var s:string):trep;overload;

opt entspricht von=0 und bis=1, d.h. das Pattern ist optional.

NON

Auch eine Verneinung ist möglich. Die Klasse **tnon** matcht den leeren String, wenn an der aktuellen Position ein Pattern nicht gematcht werden kann.

function non(apat:tpattern):tnon;overload;
function non(apat:array of const):tnon;overload;
function non(astr:string):tnon;overload;

Beispiel:
pat([non(['xyz',alt([notany(bu),rpos(0)])]),anyspan(bu)])
matcht jede Buchstabenfolge mit Ausnahme xyz, aber doch etwa xyza.

LOOK

Ferner kann man ein Pattern auch nur dann matchen, wenn dahinter eine anderes (Lookahead)-Pattern folgt. Das zu matchende Pattern kann auch leer sein, dann wird jeder String gematcht, hinter dem das Lookahead-Pattern folgt. Erkannt wird der String von der aktuellen Position bis zum ersten Auftreten des Lookahead Pattern, jedoch ohne dieses.
Bei Look gibt es keine Alternativen.
Wenn anchor=true ist, muss das Pattern an der aktuellen Position sofort kommen, sonst können vorher noch beliebige andere Zeichen kommen.

function look(alpat:tpattern):tlook;overload;
function look(alpat:tpattern;var s:string):tlook;overload;
matcht den String bis alpat

function look(apat,alpat:tpattern;aanchor:boolean):tlook;overload;
function look(apat,alpat:tpattern;aanchor:boolean;var s:string):tlook;overload;
matcht apat wenn es gefolgt ist von alpat. Zurück geliefert wird String bis alpat
Bei aanchor=false können vor apat noch beliebige Zeichen kommen

FIND, FINDWITH

Ebenfalls nach einem Pattern sucht die Klasse tfind. Zusätzlich können für den String bis zum gesuchten Pattern Klammer- und Quotebedingungen angegeben werde, wie bei tsbal, gematcht wird der String bis zum ersten Auftreten des Pattern, jedoch wieder ohne dieses.
Ist der erste Parameter ein String wird bis zum Auftreten des String gematcht.
find('A') entspricht **arb,find('A')** jedoch ohne backtracking
Auch bei find gibt es keine Alternativen.
function find(apat:tpattern):tfind;overload;
function find(apat:tpattern;var s:string):tfind;overload;
function find(apat:tpattern;aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tfind;overload;
function find(apat:tpattern;aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char;var s:string):tfind;overload;

Es wird das Pattern **apat** gesucht. Auf dem Weg dahin müssen die Pattern aus **aklaauf** und **aklazu** jeweils balanciert vorkommen. Das n-te Pattern **aklaauf** bezieht sich auf das n-te Pattern in **aklazu.** Balanciert bedeutet, dass nie mehr Ende-Pattern vorkommen dürfen als Anfang-Pattern. Ferner muss am Ende die Anzahl gleich sein.
Das gesuchte Pattern, z.B. darf jedoch dann nicht berücksichtigt werden, wenn es innerhalb eines Strings vorkommt. Dies wird durch die Pattern **aquoteauf,aquotezu** spezifiziert. Nach einem Pattern aus **aquoteauf** wird nicht weiter erkannt, bis das entsprechende Pattern aus **aquotezu** vorgekommen ist. Bei den Quote-Pattern wird jedoch keine Klammerstruktur berücksichtigt, daher können für Quoteanfang und Quoteende auch die gleichen Zeichen genommen werden, etwa ' oder " bei Strings.
Zusätzlich kann ein Zeichen **aslash** definiert werden, hinter dem keine Klammer- und Quotezeichen erkannt werden. Soll kein Slash-Zeichen benutzt werden, gibt man #0 an.

Um nicht immer die komplette Parameterlist für find angeben zu müssen, gibt es eine Reihe von überlagerten Aufrufen, die intern auf den allgemeinen Aufruf zurückgeführt werden.

function findb(apat:tpattern;const klastr:string):tfind;overload;
function findb(apat:tpattern;const klastr:string;var s:string):tfind;overload;

Ist die Klammerstruktur durch Zeichen oder Zeichenfolgen gegeben, kann dies mit einem String spezifiziert werden.
Beispiel:**';(;);[;];{;})'** als klastr liefert Klammerpaare () [] und {} 1. Zeichen ist Trenner

function findbq(apat:tpattern;const klastr,quotestr:string):tfind;overload;
function findbq(apat:tpattern;const klastr,quotestr:string;var s:string):tfind;overload;
function findbq(apat:tpattern;const klastr,quotestr:string;aslash:char):tfind;overload;
function findbq(apat:tpattern;const klastr,quotestr:string;aslash:char;var s:string):tfind;overload;

function findq(apat:tpattern;const quotestr:string):tfind;overload;
function findq(apat:tpattern;const quotestr:string;var s:string):tfind;overload;
function findq(apat:tpattern;const quotestr:string;aslash:char):tfind;overload;
function findq(apat:tpattern;const quotestr:string;aslash:char;var s:string):tfind;overload;

Sind die Quotes durch Zeichen definiert, kann das durch einen String angegeben werden.
Beispiel: find(';{;};(*;*)}') liefert die Paare { } und (* *) 1. Zeichen ist Trenner
Statt Pattern können bei find auch String gesucht werden.

function find(const astr:string):tfind;overload;
function find(const astr:string;var s:string):tfind;overload;

function findb(const astr:string;const klastr:string):tfind;overload;
function findb(const astr:string;const klastr:string;var s:string):tfind;overload;

function findbq(const astr:string;const klastr,quotestr:string):tfind;overload;
function findbq(const astr:string;const klastr,quotestr:string;var s:string):tfind;overload;

function findbq(const astr:string;const klastr,quotestr:string;aslash:char) :tfind; overload;
function findbq(const astr:string;const klastr,quotestr:string;aslash:char;var s:string) :tfind;overload;

function findq(const astr:string;const quotestr:string):tfind;overload;
function findq(const astr:string;const quotestr:string;var s:string):tfind;overload;
Der String bis zum String astr muss die Quotebedingungen erfüllen.

function findq(const astr:string;const quotestr:string;aslash:char):tfind;overload;
function findq(const astr:string;const quotestr:string;aslash:char;var s:string) :tfind;overload;
Zusätzlich kann ein Zeichen definiert werden, hinter dem keine Klammer- und Quotezeichen erkannt werden.

Ebenfalls nach einem Pattern sucht die von tfind abgeleitete Klasse tfindwith.
Hier wird jedoch nicht nur der String bis zum Pattern gematcht sondern auch das gesuchte Pattern.
Gematcht wird der String bis zum ersten Auftreten des Patterns, jedoch diesmal zusätzlich inklusive des Patterns.
Ist der erste Parameter ein String wird bis zum Auftreten des String und der String gematcht.
Der String bis zum Pattern und das Pattern werden in verschiedenen Variablen zurück geliefert.
Auch bei findwith gibt es keine Alternativen.

function findwith(apat:tpattern):tfindwith;overload;
function findwith(apat:tpattern;var s,spat:string):tfindwith;overload;
in s wird der String bis zum Pattern und in spat das Pattern geliefert.

function findwith(apat:tpattern;aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tfindwith;overload;
aklaauf und aklazu enthalten die Klammer-Pattern, aquoteauf und aquotezu die Quote Anfangs- und Endpattern, aslash enthält ein Zeichen, das das nachfolgende Zeichen quotet.

function findwith(apat:tpattern;aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char;var s,spat:string):tfindwith;overload;
in s wird der String bis zum Pattern und in spat das Pattern geliefert.

function findwithb(apat:tpattern;const klastr:string):tfindwith;overload;
function findwithb(apat:tpattern;const klastr:string;var s,spat:string):tfindwith;overload;
Beispiel:findwithb(apat,';(;[;{;};];)') liefert Klammerpaare () [] und {} 1. Zeichen ist Trenner

function findwithbq(apat:tpattern;const klastr,quotestr:string):tfindwith;overload;
function findwithbq(apat:tpattern;const klastr,quotestr:string;var s,spat:string) :tfindwith; overload;
Beispiel:findwithbq(apat,';(;[;{;};];)','','','','') erkennt
die Klammerpaare () [] und {} (1. Zeichen ist Trenner) und '…' bzw. "…" als String

function findwithbq(apat:tpattern;const klastr,quotestr:string;aslash:char):tfindwith;overload;

function findwithbq(apat:tpattern;const klastr,quotestr:string;aslash:char;var s,spat:string):tfindwith; overload;

function findwithq(apat:tpattern;const quotestr:string):tfindwith;overload;
function findwithq(apat:tpattern;const quotestr:string;var s,spat:string):tfindwith;overload;
Beispiel:findwithq（apat,'','','',""）erkennt die '…' bzw. "…" als String in denen apat nicht erkannt wird.

function findwithq(apat:tpattern;const quotestr:string;aslash:char):tfindwith;overload;
function findwithq(apat:tpattern;const quotestr:string;aslash:char;var s,spat:string):tfindwith; overload;

function findwith(const astr:string):tfindwith;overload;
function findwith(const astr:string;var s,spat:string):tfindwith;overload;
Erkennt den String bis astr und astr
in s wird der String bis zum Pattern und in spat das Pattern geliefert.

function findwithb(const astr:string;const klastr:string;var s,spat:string):tfindwith; overload;
Erkennt astr wenn bis dahin die Klammerstruktur stimmt.

function findwithbq(const astr:string;const klastr,quotestr:string):tfindwith;overload;
function findwithbq(const astr:string;const klastr,quotestr:string;var s,spat:string) :tfindwith; overload;
Erkennt astr wenn bis dahin die Klammerstruktur und die Quotestruktur stimmt.
function findwithbq(const astr:string;const klastr,quotestr:string;aslash:char):tfindwith; overload;
function findwithbq(const astr:string;const klastr,quotestr:string;aslash:char;var s,spat:string): tfindwith; overload;
Zusätzlich kann auch ein Quote-Zeichen definiert werden.

function findwithq(const astr:string;const quotestr:string):tfindwith;overload;
function findwithq(const astr:string;const quotestr:string;var s,spat:string):tfindwith; overload;
Erkennt astr wenn bis dahin die Quotestruktur stimmt.

function findwithq(const astr:string;const quotestr:string;aslash:char):tfindwith;overload;
function findwithq(const astr:string;const quotestr:string;aslash:char;var s,spat:string) :tfindwith; overload;
Das gleiche wieder mit Quotezeichen aslash,

(4) Spezielle Pattern

Um den Patternmatching-Algorithmus abzubrechen bzw. weitere Alternativen zu suchen gibt es einige spezielle Pattern.

ABORT

Kommt der Algorithmus an eine Instanz von tabort bricht er sofort ohne Erfolg ab.
function abort:tabort;

FAIL

Möchte man nachdem eine Patternfolge erfolgreich gematcht wurde, weitere Alternativen finden, kann man eine Instanz von tfail hinzufügen. Hier liefert matchnext immer false, sodass in den vorhergehenden Pattern Alternativen gesucht werden.
function fail:tfail;

FENCE

Umgekehrt, kann die Suche nach Alternativen durch eine Instanz von tfence verhindert werden. Beim normalen Durchlauf von links nach rechts matcht tfence den Leerstring. Beim Backtracking matcht tfence jedoch nicht und verhindert zusätzlich, dass in den Pattern davor nach Alternativen gesucht wird.
function fence:tfence;

CRLF

Hat man einen String, der aus mehreren Zeilen besteht, kann man mit tcrlf den String zeilenweise bearbeiten
function crlf:tcrlf;
tcrlf matcht die Folge CR/LF (#13#10) falls vorhanden oder die Zeichen CR(#13) oder LF(#10).

WORDSTART, WORDEND

function wordstart:twordstart;
Matcht den Leerstring, wenn sich der Cursor am Wortanfang befindet d.h. hinter dem Cursor ein Zeichen aus namechars und davor ein Zeichen nicht aus namechars

function wordend:twordend;
Matcht den Leerstring, wenn sich der Cursor am Wortende befindet d.h. vor dem Cursor ein Zeichen aus namechars und dahinter ein Zeichen nicht aus namechars

REPL

Mit der Klasse trepl kann man ein Pattern nicht nur matchen, sondern bei erfolgreichem Match auch den gematchten String durch einen anderen ersetzen. Eine Instanz von trepl hat allerdings keine Alternative, da der zu matchende String bereits ersetzt wurde. Werden in Pattern links davon nach Alternativen gesucht, wird beim neuen Match mit dem geänderten String gearbeitet.

function repl(apat:tpattern;asub:tsv):trepl;overload;
function repl(apat:array of const;asub:tsv):trepl;overload;
function repl(apat:array of const;asub:array of const):trepl;overload;
function repl(apat:tpattern;asub:array of const):trepl;overload;
Der ersetzende String ist eine Instanz von tsv. D.h. es kann eine Stringvariable sein, die erst beim Matching ihren Wert erhält. Der ersetzende String kann sich auch aus mehreren Teilen zusammensetzen, die jeweils Instanzen von tsv sein können.

(5) Aufrufe der Funktionen zum Patternmatching

MAT

function mat(var s:string;ab:integer;const pat:tpattern):boolean;overload
Test: s matcht das Pattern pat irgendwo ab ab. Bei anchorglb=true muss genau ab ab gematcht werden

function mat(var s:string;ab:integer;const pat:tpattern;var apos,alen:integer):boolean;

overload;
Test: s matcht pat irgendwo ab ab wenn ja apos mit der Position und alen mit der Länge des gematchten Strings setzen. Bei anchorglb=true muss genau ab ab gematcht werden

function mat(var s:string;ab:integer;const pat:tpattern;var apos:integer;var s2:string) :boolean;overload;
Wie oben aber s2 enthält den gematchten String

function mat(var s:string;ab:integer;par:array of const):boolean;overload;
function mat(var s:string;ab:integer;par:array of const;var apos,alen:integer):boolean; overload;
function mat(var s:string;ab:integer;par:array of const;var apos:integer;var s2:string) :boolean;overload;
Wie oben, das Pattern par ist eine sequentielle Folge von Pattern.

function mat(var s:string;ab:integer;const s2:string;var apos:integer):boolean; overload;
Test: s matcht den String s2 irgendwo ab ab wenn ja apos mit der Position es gematchten String setzen. Bei anchorglb=true muss genau ab ab gematcht werden

function mat(var s:string;const pat:tpattern):boolean;overload
Test: s matcht pat irgendwo (ab=0) Bei anchorglb=true muss genau am Anfang gematcht werden

function mat(var s:string;const pat:tpattern;var apos,alen:integer):boolean;overload;
Test: s matcht pat irgendwo (ab=0) wenn ja apos mit der Position und alen mit der Länge des gematchten String setzen. Bei anchorglb=true muss genau am Anfang gematcht werden

function mat(var s:string;const pat:tpattern;var apos:integer;var s2:string):boolean; overload;
Wie oben aber s2 enthält den gematchten String

function mat(var s:string;par:array of const):boolean;overload;
function mat(var s:string;par:array of const;var apos,alen:integer):boolean;overload;
function mat(var s:string;par:array of const;var apos:integer;var s2:string):boolean; overload;
Wie oben, das Pattern par ist eine sequentielle Folge von Pattern.

function mat(var s:string;const s2:string;var apos:integer):boolean;overload;
Test: s matcht den String s2 irgendwo(ab=0) wenn ja apos mit der Position es gematchten String setzen. Bei anchorglb=true muss genau am Anfang gematcht werden

REPLACE

Mit der Funktion replace wird bei erfolgreichem Match ein String geliefert, bei dem der gematchte Teilstring durch einen neuen String ersetzt wird.

function replace(const s:string;pat:tpattern;newstr:tstri;rep:boolean=false; rek:boolean=false; ancho:boolean=false;ab:integer=0):string;overload;
Im String s wird der durch das Pattern pat gematchte Teilstring durch den von der Instanz newstr von tsv definierten String ersetzt. In newstr können Stringvariablen vorkommen, die erst bei dem Matching von pat ihren Wert erhalten haben. Insbesondere kann newstr eine Instanz von tcat sein, d.h. sich aus mehreren Teilen zusammensetzen.
Funktionswert ist der String s mit den Ersetzungen, wenn pat gematcht wurde.
Ist rep=true, wird versucht im restlichen Teilstring von s einen weiteren Match von pat zu finden. Solange dies erfolgreich ist, werden auch diese matchenden Teilstring von s durch newstr ersetzt.

Ist rek=true wird versucht das gleiche replace auf den sich aus newstr ergebenden ersetzenden String wieder anzuwenden, bis pat nicht mehr gematcht wird.

function replace(const s:string;arr:array of const;newstr:tstri; rep:boolean=false; rek:boolean=false; ancho:boolean=false;ab:integer=0):string;overload;
Analog das Pattern wird durch die Patternfolge arr gebildet.

function replace(const s:string;pat:tpattern;newstr:array of const; rep:boolean=false; rek:boolean=false;ancho:boolean=false;ab:integer=0):string;overload;
Analog der ersetzende String wird durch das Stringfeld newstr gebildet.

function replace(const s:string;arr:array of const;newstr:array of const; rep:boolean=false; rek:boolean=false; ancho:boolean=false;ab:integer=0):string; overload;
Analog das Pattern wird durch die Patternfolge arr gebildet. Der ersetzende String wird durch das Stringfeld newstr gebildet.
function replace(const s:string;pat:tpattern;newstr:string; rep:boolean=false; rek:boolean=false; ancho:boolean=false;ab:integer=0):string;overload;
function replace(const s:string;arr:array of const;newstr:string;rep:boolean=false; rek:boolean=false;ancho:boolean=false;ab:integer=0):string;overload;
Analog, der gematchte Teil wird duch den String newstr ersetzt.

Analog wie replace jedoch ohne zu ersetzenden String arbeitet die Funktion match.
function match(s:string;pat:tpattern;rep:boolean=false;rek:boolean=false; ancho:boolean=false; trenner:string;ab:integer=0):string; overload;
function match(s:string;arr:array of const;rep:boolean=false;rek:boolean=false; ancho:boolean=false;trenner:string;ab:integer=0):string;overload;
Das Ergebnis ist der bzw. die gematchten Strings. Sie werden jeweils durch den Trenner getrennt.
Beispiel:
s:='((a)((b)(c))(d))';
s2:=match(s,bal(',(,)'),true,true,false,';');
liefert für s2:
((a)((b)(c))(d));(a);((b)(c));(b);(c);(d)
d.h eine Liste aller Teilstring von s, die bezüglich der Klammern balanciert sind.

Möchte man alle Vorkommen eines Pattern in einem String ermitteln, kann man auch die Funktion matchall benutzen.
function matchall(var s:string;ab:integer;pat:tpattern;var apos:arrayofinteger; var sl:arrayofstring) : integer;overload;
In s werden alle Teilstrings, die pat matchen ermittelt; apos liefert die Positionen und sl die gematchten Strings. Der Funktionswert ist die Anzahl der Ersetzungen.
Beispiel: Sei name ein Pattern, das Namen erkennt so liefert
matchall(ss,0,name,aposl,sl)
alle Namen die in ss vorkommen in dem Stringarray sl und die Position in dem Integerarray apos;
Die Parameter apos und sl sind Dynamische Arrays. Als formaler Parameter muss in Delphi bei dynamischen Arrays jeweils ein Bezeichner genommen werden, um dynamische arrays von offenen Array-Parametern zu unterscheiden. Daher wurden folgende Typen definiert:
type arrayofinteger=array of integer;
arrayofstring=array of string;

function matchall(s:string;ab:integer;pat:array of const;var apos:arrayofinteger;var sl:arrayofstring):integer;overload;
Analog mit der Patern-folge pat

function matchall(s:string;ab:integer;pat:tpattern;var sl:arrayofstring):integer;

overload;
In s werden alle Teilstring, die pat matchen ermittelt; sl liefert die gematchten String
Beispiel: matchall(ss,0,name,sl) liefert alle Namen in ss

function matchall(s:string;ab:integer;pat:array of const;var sl:arrayofstring):integer; overload;
Analog mit der Paternfolge pat

function matchall(s:string;pat:tpattern;var apos:arrayofinteger;var sl:arrayofstring):integer; overload;
function matchall(s:string;pat:array of const;var apos:arrayofinteger;var sl:arrayofstring) :integer;overload;
function matchall(s:string;pat:tpattern;var sl:arrayofstring):integer;overload;
function matchall(s:string;pat:array of const;var sl:arrayofstring):integer;overload;
Analog wie vorher, jedoch immer ab Position 0

Will man alle Sätze einer Datei bearbeiten, kann man die Prozedur **substitutefile** benutzen.

procedure substitutefile(const quelle,ziel:string;arr,newstr:array of const; rep:boolean=false;rek:boolean=false;ancho:boolean=false;ab:integer=0);overload;
In der Datei quelle wird in jeder Zeile s die Funktion replace(s,arr,newstr,rep,rek,ancho,ab) aufgerufen. Der Funktionswert bildet die Zeile in der Datei ziel.

Die Filterfunktion für eine Datei, kann auch allgemein mit einer Prozedur von folgendem Typ ausgeführt werden:
type tsubprocedure=procedure(const s1:string;var s2:string);
s1 ist der Eingabestring und s2 der Ausgabestring.
Dies kann dann benutzt werden in der Prozedur **filefilter.**
procedure filefilter(const quelle,ziel:string;proc:tsubprocedure);
Auf jeden Satz der Datei quelle wird die Prozedur proc angewandt und das Ergebnis in die Datei ziel geschrieben.
Möchte man die Erkennung über die Zeilengrenzen hinweg machen, kann auch der gesamte Text gescannt werden. Dann benutzt man:
procedure textfilter(const quelle,ziel:string;proc:tsubprocedure);
Alle Zeilen werden mit jeweils CR/LF zu einem String verknüpft. Auf diesen String wird die Prozedur proc angewandt;

Sollen alle Sätze einer Datei nur bearbeitet werden, so kann die Prozedur **filetest** benutzt werden.
procedure filetest(const quelle,ziel:string;proc:tsubprocedure);overload;
procedure filetest(const quelle,ziel:string;proc:tsubprocedure;mitzeilennr:boolean); overload;
Auf jeden Satz der Datei quelle wird die Prozedur proc angewandt. Ist der Ausgabe String s2 von proc nicht leer, wird er ausgegeben. Optional kann noch die Ausgabe der Satznummer bewirkt werden.

(6) Vordefinierte Mengen und Pattern
const letters:tcharset=['A'..'Z','a'..'z','Ä','Ö','Ü','ä','ö','ü','ß'];
Alle Buchstaben
const digits:tcharset=['0'..'9'];
Alle Ziffern
namechars:tcharset=['0'..'9','A'..'Z','a'..'z','Ä','Ö','Ü','ä','ö','ü','ß','_'];
Alle Buchstaben und Ziffern

var name:tpattern= pat([any(letters),span(namechars)]);

Alle Bezeichner
var blanks:tpattern=span(' ');
Längste Folge von Blanks
var number:=pat([any(digits),span(digits)]);
Alle Integer Zahlen

(7) Anwendungsbeispiele

(7.1) Entfernen der Kommentare in einem Pascal-Programm
delcomment.pas

In einem Pascal-Programm sollen alle Kommentare entfernt werden. Kommentare werde gebildet durch die Paare { und } bzw. (* und *) bzw. // und Zeilenende. Die Kommentarzeichen haben aber keine Bedeutung, wenn sie in einem anderen Kommentar auftreten oder in einem String.

```
program delcomment;
{$APPTYPE CONSOLE}
uses SysUtils,mysnobol;
var f1,f2:text;
var s,ss,s2:string;
 i:integer;
var pcomment:tpattern;
function comm(start,ende:string):tpattern;
begin
result:=pat([start,non('$'),findwithq(ende,'''')]);
end;
function comm2(start,ende:string):tpattern;
begin
result:=pat([start,findwithq(ende,'''')]);
end;
begin
pcomment:=alt([comm('{','}'),comm('(*','*)'),comm2('//',#13#10)]);
write('Eingabedatei:');readln(s);
assign(f1,s);reset(f1);
write('Ausgabedatei:');readln(s);
assign(f2,s);rewrite(f2);
s:='';
while not eof(f1) do begin
 readln(f1,ss);s:=s+ss+#13#10;
end;
close(f1);
s2:=replace(s,pcomment,[''],true);
s2:=replace(s2,plus(#13#10),#13#10,true);
i:=pos(#13#10,s2);
while i>0 do begin
 writeln(copy(s2,1,i-1));
 writeln(f2,copy(s2,1,i-1));delete(s2,1,i+1);
 i:=pos(#13#10,s2);
end;
writeln(s2);
writeln(f2,s2);
close(f2);
write('Weiter mit CR');readln;
end.
```

function comm ist ein Beispiel für eine Funktion, die ein Pattern liefert, das von den Parametern abhängt.
Es wird die Zeichenfolge für den Start des Kommentars gesucht. Anschließend darf bei den Kommentaren mit { ... } oder (* ... ') kein **$** folgen, damit die Direktiven nicht gelöscht werden. Dies wird erreicht durch das Pattern non. Dies matcht den leeren String, wenn an dieser Stelle nicht das angegebene Pattern, in diesem Fall das **$,** vorkommt. Dann wird der String für das Ende des Kommentars gesucht. Mit findwithq(ende,"") wird sichergestellt, dass das Ende nicht in einem String vorkommt. Bei einem // Kommentar darf **$** vorkommen, daher wird hier das Pattern comm2, was das non Pattern nicht enthält, genommen.
Das Pattern pcomment enthält die verschiedenen Kommentar-alternativen. Die Funktion replace(s,pcomment,[''],true) sucht alle Vorkommen von Kommentaren und ersetzt sie durch den leeren String.

(7.2) Vertauschen der Zeilen- und Spaltenindizes in einer Matrix

array.pas

```
program parray;
// in einem Array wird der erste mit dem zweiten Index vertauscht
{$APPTYPE CONSOLE}
uses mysnobol;
var st:string;
var s,ss:string;
 s1,s2,s3,s4:string;
begin
s:='a[2] b[3,z[4]] c d[1,x[2,3],y[1,2,3]]';
writeln(s);
ss:=replace(s,pat(['[',findwithb(',',';
[;]',s1,st),findwithb(any(',]',st),';[;]',s2,s3)]),
['[',sv(s2),',',sv(s1),sv(s3)],true,true);
writeln(ss);
readln;
end.
```

Das Pattern

```
pat(['[',findwithb(',',';[;]',s1,st),findwithb(any(',]',st),';
[;]',s2,s3)])
```

findet den ersten und den zweiten Indexausdruck in einem Array.
Auf s1 wird der erste und auf s2 der zweite Indexausdruck zurück geliefert. In s3 wird das Endzeichen, entweder] oder , geliefert.
Durch den Aufruf von replace wird dieses erkannt und im ersetzten String werden die beiden Indexausdrücke vertauscht. Durch die beiden letzten Parameter wird dies wiederholt und rekursiv gemacht.
Man erhält die Ausgabe:

```
d[x[2,3,4],y]
d[y,x[3,2,4]]
a[2] b[3,z[4]] c d[1,x[2,3],y[1,2,3]]
a[2] b[z[4],3] c d[x[3,2],1,y[2,1,3]]
```

Am zweiten Beispiel sieht man, wie die Vertauschung auch rekursiv angewandt wird. Dies ist mit Regulären Ausdrücken, wie in Perl ohne eine spezielle Programmierung eines Stacks nicht möglich.

(7.3) Erkennen einer XML-Struktur

xml.pas

Beim Erkennen der XML-Struktur eines Dokuments muss man zu einem Start-Tag z.B. <a> das entsprechende Ende-Tag finden. Dazwischen können aber weitere Vorkommen von <a> mit entsprechenden vorkommen. Beim Erkennen des Starttags wird ein String erkannt, der in das Pattern zum Erkennen des Endtags eingehen muss. Ferner muss die balancierte Klammerstruktur erkannt werden.
Um das Starttag zu erkennen wird folgendes Pattern benutzt:

```
function start(var s:string):tpattern;
begin
result:=pat(['<',pat(name,s),findq('>','"'),'>']);
end;
```

Der vom Pattern name erkannte Tagname wird auf s abgespeichert. Das Tag wird beendet mit >, jedoch nicht in einem String. Das Pattern soll jedoch nicht die Tags ohne Endtag erkennen z.B. <a/>. Deswegen wird hierfür ein gesondertes Pattern definiert.

```
function startende(var s:string):tpattern;
begin
result:=pat(['<',findq('/>','"'),'/>'],s);
end;
```

Tritt der erkannte Starttag wieder auf muss dies für die Ermittlung der Balance erkannt werden.

```
function start2(var s:string):tpattern;
begin
result:=pat(['<',sv(s),findq('>','"'),'>']);
end;
```

Das Pattern für den Endtag ist analog aufgebaut:

```
function ende(var s:string):tpattern;
begin
result:=pat(['</',sv(s),'>']);
end;
```

Diese Pattern werden dann in der Testroutine benutzt.

```
procedure test(stufe:string;s:string);
var s1,s2,s3,s4,s5:string;
var ipos,apos:integer;
var res:boolean;
var n:integer;
var stufe2:string;
begin
if s='' then exit;
ipos:=0;
n:=1;
//trace(2);
repeat
 s1:='';s2:='';s3:='';s4:='';s5:='';
 res:=mat(s,ipos,[findq('<',quotestr),
   alt([pat([start(s1),
        findwith(ende(s1),[start2(s1)],[ende(s1)],
         ['"','''''],['"','''''],#0,s3,s4)]),
        startende(s5)])],
   apos,s2);
 if res then begin
  stufe2:=stufe+'.'+inttostr(n);
  writeln(outp,stufe2);
```

```
  writeln(outp,s3);
  writeln(outp,'----------------------');
  writeln(stufe2);
  writeln(s3);
  writeln('----------------------');
  if (s5='')and(s3<>'')then test(stufe2,s3);
  ipos:=apos+length(s2);
  inc(n);
 end;
until not res;
end;
```

Erkannt wird immer entweder ein StartEndeTag oder ein Starttag, dann eine balancierte Folge von diesem Starttag und dem entsprechenden Endtag, gefolgt von dem Endtag.

Entscheidend ist folgendes Pattern:

```
findwith(ende(s1),[start2(s1)],[ende(s1)],
         ['"',''''],['"',''''],#0,s3,s4)),
```

Es wird das entsprechende Endetag (`ende(s1)`) gesucht. Dies wird aber nur dann erkannt, wenn bis dahin das entsprechende Starttag und Endetag balanciert sind, was durch `[start2(s1)],[ende(s1)]` erreicht wird.
Auch darf das Tag nicht in einem String vorkommen, was durch `['"',''''],['"',''''] ` als Quoteangaben sichergestellt wird.
Die Benutzung von **`findwith`** hat den Vorteil, dass nicht so viele Alternativen durchsucht werden müssen. In der Prozedur test sind die einzigen Alternative, die beiden den dem **alt** Pattern.
Das gleiche kann man auch erreichen mit dem SNOBOL Pattern **arb** .

```
res:=mat(s,ipos,[arb,
      alt([pat([start(s1),arb(s2),
        star(balquote( [start2(s1)], [ende(s1)],[],[],#0),s4),
        arb(s5),ende(s1)]),
       startende(s6)])],apos,s3);
```

Bei **arb** wird jedoch für jede Länge neu probiert. Durch die beiden geschachtelten **arb** wird der Aufwand quadriert.

(7.3) Erkennen der Texte in einer HTML Datei

htmltexte.pas

Um z.B. von einer HTML Datei eine englische Version zu erstellen, ist es erforderlich die Texte außerhalb der Tags zu erkennen und in einer Datei mit Angabe von Position und Länge zu speichern. Diese Datei kann dann übersetzt werden und die übersetzten Texte können wieder an den entsprechenden Stellen eingefügt werden.

Dies wird erreicht mit:

```
tag:=balquote(',<,>',',","');
apos:=0;
s1:='';
while mat(s,apos,upto('<'),apos2,s2) do begin
 l:=length(s2);
 if l>0 then s1:=s1+'[['+inttostr(apos2)+'!!'+inttostr(l)+']]'+s2+#13#10;
 apos:=apos2+l;
 if not mat(s,apos,tag,apos2,l)then break;
 apos:=apos2+l;
end;
```

Ist etwa
s:='vorher 12<45>">12345 Text 6789 uvw';

so erhält man folgende Ausgabe:
```
[[0!!7]]vorher
[[24!!11]]12345 Text
[[38!!6]] 6789
[[52!!4]] uvw
```

Nach der Übersetzung kann mit den Angaben von Position und alter Länge der alte Text leicht durch den übersetzten ersetzt werden. Durch die Quote Angaben stört es auch nicht, wenn in einem Tag-Parameter ein Tagzeichen (< oder >) vorkommt.
```
matchall(s1neu,['[[',number,'!!',number,']]', look(alt(['[[',rpos(0)]))],
sl);
for i:=sl.count-1 downto 0 do begin
 ss:=sl[i];
 mat(ss,['[[',pat(number,s2),'!!',pat(number,s3),']]',rem(s4)]);
 ipos:=strtoint(s2);
 ilen:=strtoint(s3);
 if system.copy(s4,length(s4)-1,2)=#13#10then setlength(s4,length(s4)-2);
 system.delete(sneu,ipos+1,ilen);
 system.insert(s4,sneu,ipos+1);
end;
```

s1neu ist der übersetzte Text. Durch den matchall Aufruf wird er in die Einzelnen zu ersetzenden Folgen zerlegt. Die Ersetzungen müssen von hinten nach vorne vorgenommen werden, um die originalen Positionen zu erhalten. Der Pattern look matcht den String bis zum gesuchten Pattern, jedoch ohne dieses. Hier bis zum nächsten Auftreten von '[[' oder bis zum Ende des Textes. Dieser String wird auf s4 abgespeichert.
Entsprechend der Positionsangabe wird der alte Teilstring gelöscht und durch den neuen ersetzt. Zu beachten ist, dass beim SNOBOL Paket die Position des 1. Zeichen 0 ist während bei den Pascal Prozeduren die String mit dem Index 1 beginnen.

(7.4) Benutzung der Funktion sv
(dokutest.pas)

In einem String sollen alle doppelt vorkommenden Buchstaben nur noch einmal vorkommen.

s2:=replace(s,[any(letters,s1),sv(s1)],sv(s1),true);

Der durch any(letters,s1) gematchte Buchstabe wird auf s1 abgespeichert.
Bei sv(s1) wird der aktuelle Wert von s1 genommen, sodass dieses Pattern nur matcht, wenn das nachfolgende Zeichen das gleiche ist.
Die beiden gleichen Zeichen werden dann durch ein Zeichen ersetzt.
Durch den letzten Parameter, wird dies für den nachfolgenden Teilstring solange wiederholt, bis kein Zeichen mehr doppelt vorkommt. Kommt ein Buchstabe 3x vor, bleibt er zweimal übrig, da der nachfolgende Teilstring nicht mehr 2 aufeinander folgende Buchstaben enthält.
Aus dem String 'xaaabccddddz' wird 'xaabcddz'

Um alle dreifachen nach einfach zu bringen analog:
```
s2:=replace(s,[any(letters,s1),sv(s1),sv(s1)],sv(s1),true);
```
Aus dem String 'xaaabccddddz' wird 'xabccddz'

Will man alle mehrfachen nach einfach bringen kann die Funktion plus verwendet werden.
```
s2:=replace(s,[any(letters,s1),plus(sv(s1))],sv(s1),true);
```

Aus dem String 'xaaabccddddz' wird 'xabcdz'
Dazu nutzt man zweckmäßig, dass greedy=true ist, damit bei plus soviel Wiederholungen, wie möglich genommen werden.
Bei greedy=false würde man zwar das gleiche Ergebnis erhalten, allerdings erst durch mehrfaches wiederholen des Matching bis die maximal mögliche Anzahl von Wiederholungen erreicht wird.
Schaltet man die Wiederholungen durch die Funktion fence aus so wird nur die erste Variante genommen.

s2:=replace(s,[any(letters,s1),plus(sv(s1)),fence],sv(s1));

Hiermit erhält man:
xaabccddddz

(7.5)Formatieren von HTML Texten
htmlform.pas

Es soll angenommen werden, dass eine korrekte HTML Datei vorliegt. Diese soll so formatiert werden, dass bei jedem Tag um eine Spalte eingerückt wird und beim entsprechenden Endtag wieder zurückgegangen wird. Dabei sollen auch Kommentare korrekt erkannt werden.

Obwohl die HTML Datei eine Klammerstruktur bildet, kommt man für das Formatieren mit regulären Ausdrücken aus. Beim Starttag wird eingerückt und beim Endtag wieder ausgerückt. Man muss allerdings erkennen, wenn kein getrennter Endtag auftritt. Dies ist in HTML der Fall bei bestimmten Tags wie
,<meta> etc. Ferner gibt es bei <!... und <?... kein Endtag. Auch die HTML Kommentare, die mit <!-- beginnen und mit --> enden müssen erkannt werden.

```
program htmlform;
{$APPTYPE CONSOLE}
uses classes,strutils,sysutils,mystring,mysnobol;
var f1,outp:text;
var eingabe:string='index.html'; //Name der Eingabedatei
 ausgabe:string='';//'out2.txt';        //Name der Ausgabedatei

procedure mywrite(ss:string;level:integer);
var bl,s2:string;
 i:integer;
 sl:arrayofstring;
begin
bl:=dupestring(' ',level);
sl:=tokenize(ss,#13#10);
for i:=0 to length(sl)-1 do begin
 s2:=trim(sl[i]);
 if s2<>''then begin
  s2:=bl+s2;
  writeln(outp,s2);
 end;
end;
end;

const nohne=13;
const ohneendtag:array[0..nohne-1]of string=
('area','base','basefont','br','col','frame','hr','img','input','isin-
dex',
 'link','meta','param');
```

```
function isohneendtag(s:string):boolean;
// Prüfen ob Tag ohne Endtag bei Html vorliegt
var i:integer;
begin
result:=false;
for i:=0 to nohne-1 do if s=ohneendtag[i]then begin
 result:=true;break;
end;
end;

const quotestr=';'';'';";"';
var level:integer;

function start(var s,rest:string):tpattern;
var s2:string;
 res:tpattern;
begin
res:=pat(['<',pat(name,s),findwithq('>',quotestr,rest,s2)]);
result:=res;
end;

procedure test(s:string);
var s1,s2,s3:string;
 ipos,apos:integer;
 res:boolean;
begin
ipos:=0;
repeat
 s1:=' ';s2:=' ';s3:=' ';
 res:=mat(s,ipos,findq('<',quotestr),apos,s1);
 if not res then begin
  s1:=copy(s,ipos+1,100000);
  mywrite(s1,level);
  break;
 end;
 mywrite(s1,level);
 ipos:=ipos+length(s1);
 // Kommmentar
 res:=mat(s,ipos,['<!--',findwithq('-->',quotestr,s2,s3)],s1);
 if res then begin
  if pos(#13#10,s1)>0 then begin
   mywrite('<!--',level);
   mywrite(s2,level+1);
   mywrite('-->',level);
  end else mywrite(s1,level);
  ipos:=ipos+length(s1);
 end else begin
  // <! .... >
  res:=mat(s,ipos,['<',alt(['!','?']),findwithq('>',quotestr)],s1);
  if res then begin
   mywrite(s1,level);
   ipos:=ipos+length(s1);
  end else begin
   res:=mat(s,ipos,start(s1,s3),apos,s2);
```

```
   if res then begin
    ipos:=ipos+length(s2);
    mywrite(s2,level);
    if (s3<>'')and(s3[length(s3)]='/')then begin
     // <tag/>
    end else begin
     // <tag>
     if not isohneendtag(s1) then level:=level+1;
    end;
   end else begin
    res:=mat(s,ipos,['</',findwithq('>',quotestr)],s1);
    if res then begin
     level:=level-1;
     mywrite(s1,level);
     ipos:=ipos+length(s1);
    end else mywrite('++++Fehler:'+copy(s,ipos,20),0);
   end;
  end;
 end;
until false;
end;
var s,ss:string;
begin
anchor(1);
// Name der Eingabedatei
if paramcount>=1 then eingabe:=paramstr(1)
else begin
 write('Eingabedatei:');
 //readln(eingabe);
 eingabe:='test.html';
end;
if paramcount>=2 then ausgabe:=paramstr(2)
else begin
 ausgabe:='';
end;
assign(outp,ausgabe);
rewrite(outp);
assign(f1,eingabe);
reset(f1);
ss:='';s:='';
while not eof(f1) do begin
 readln(f1,s);
 ss:=ss+s+#13#10;
end;
level:=0;
test(ss);
close(f1);
close(outp);
writeln('Ende mit CR');
readln;
end.
```

Da die Erkennung ohne die Funktion bal erfolgt, kann man das gleiche auch in einer Sprache programmieren, die nur reguläre Strukturen erkannt, wie etwa mit Regex bei PERL.

Der entsprechende Algorithmus in PERL ist:

```
sub writeln{
 #return;
 my($s)= @_;
 print $s;
 print "\n";
}
sub trim($){
 my $string = shift;
 $string =~ s/^\s+//;
 $string =~ s/\s+$//;
 return $string;
}
sub mywrite{
# Ausgabe von $sa mit der Eintückung $level
# Zeilen werden an \n jeweils aufgetrennt und eingerückt
 my($sa, $level) = @_;
 my $blanks=' ' x $level;
 my $s2;
 if ($sa eq '') {return}
 do {
  if ($sa =~ /\n/) {
   $s2=$`;
   $sa=substr($sa,length($s2)+1);
  } else {
   $s2=$sa;$sa='';
  }
  $s2=&trim($s2);
  if ($s2 ne ''){
   $s2=$blanks.$s2;
   print AUSGABE "$s2\n";
  }
 } until ($sa eq '');
}

# Die Tags die bei html kein entsprechendes Endtag haben
my
@ohneendtag=('area','base','basefont','br','col','frame','hr','img','inpu
t','isindex','link','meta','param');

 sub isohneendtag{
 my($s)= @_;
 # Prüfen ob Tag ohne Endtag bei Html vorliegt
 foreach $i (@ohneendtag) {
  if ($i eq $s){
   return 1;
  }
 }
 return 0;
}

$ss=''; #Der noch zu bearbeitende String
sub find($){
 # String finden aber nicht in "..." oder '....'
```

```
 # Ergebnis  String bis zum Suchstring
 my $s=shift;
 my $sf='';
 do {
  $res1= $ss =~ /\'|\"/; #Stringanfang
  if ($res1) {
   $vor1 =$`;
   $match1 = $&;
  } else {
   $vor1=$ss;
  }
  $res2= $ss =~ /$s/; #Suchstring
  if ($res2) {
   $vor2 =$`;
   $match2 = $&;
  } else {
   $vor2 =$ss;
  }
  if (!($res1 || $res2)){ #weder ' oder " noch Suchstring
   $vor1= $ss;
   $ss='';
   return $vor1;
  }
  if (length($vor1)<length($vor2)){
   # zunächst ein String
   $sf=$sf.$vor1.$match1;
   $ss=substr($ss,length($vor1)+length($match1));
   # Ende des Strings suchen
   $res1= $ss =~ /$match1/;
   $vor1 =$`;
   $match1 = $&;
   $sf=$sf.$vor1.$match1;
   $ss=substr($ss,length($vor1)+length($match1));
  } else {
   # es folgt der Suchstring
   $sf=$sf.$vor2;
   $ss=substr($ss,length($vor2));
   return $sf; #String bis zum Suchstring
  }
 } until ($ss eq '');
}

sub test{
 $level=0;
 my $s1="";my $s2="";my $s3="";
 do {
  $bis=&find('\<');
  &mywrite($bis,$level);
  if (substr($ss,0,4) eq '<!--'){
   # Kommentar gefunden
   &mywrite('<!--',$level);
   $ss=substr($ss,4);
   $ss =~ /-->/; # Ende Kommentar
   $comment=$`;
   $ss=substr($ss,length($comment)+3);
```

```
   &mywrite($comment,$level+1);
   &mywrite('-->',$level);
  } elsif ($ss ne "") {
   $tag=&find('\>'); # Ende des Tags
   $tag=$tag.'>';
   $ss=substr($ss,1);
   if ($tag =~ /\A<\//){
    # Endtag gefunden
    $level=$level-1;
    &mywrite($tag,$level);
   } elsif ($tag =~ /\A<(!|\?)/){
    # <! oder <?
    &mywrite($tag,$level);
   } else {
    $tag =~ /\A\<(\w+?\b)/; # Name des Tags nach $1
    &mywrite($tag,$level);
    if (! &isohneendtag($1)) {$level=$level+1;}
   }
  }
 } until ($ss eq "");
}

($in,$out)=@ARGV;
if ($in eq ''){
 print('Eingabedatei:');$in=<STDIN>;
 $in=&trim($in);
 if ($in eq ''){
  $in='con.txt';
 }
}
if ($out eq ''){
 print('Ausgabedatei:');$out=<STDIN>;
 $out=&trim($out);
 if ($out eq ''){
  $out='con.txt';
 }
}

print "Formatieren $in nach $out\n";
if (!open EINGABE,"<$in"){print("$in nicht vorhanden");exit;}
if (!open AUSGABE,">$out"){print("$out kann nicht geöffnet
werden");exit;}
$ss='';
while (<EINGABE>) {$ss=$ss.$_."\n";}
&test($ss);
print("Ende mit CR\n");
$line = <STDIN>;
```

(7.6) Formatieren eines Pascal Quelltextes

(pasformat.pas)

Eine Pascal Quelle wird so formatiert, dass bei jedem begin oder repeat eine neue Zeile mit einer Einrückung erzeugt wird. Bei end oder until wird wieder eine neue Zeile erzeugt und die Einrückung wieder zurück genommen.

```
program pasformat;
{$APPTYPE CONSOLE}
uses
  sysutils,strutils,myobject,mystring,mysnobol;
const debug=false;
function strword(s:string):tpat;
// Erkennt String s, wenn er als Wort vorkommt
begin
result:=pat([wordstart,s,wordend]);
end;
const pascalkomm:string=',{,},(*,*)';
 pascalstring:string=',',''';
type tstringlist=record
 a:array[0..1000]of string;
 count:integer;
end;
procedure add(var sl:tstringlist;s:string);
begin
sl.a[sl.count]:=s;
inc(sl.count);
end;
function comm(start,ende:string;var s:string):tpattern;
// Kommentar mit Anfang start und Ende ende unter Berücksichtigung von
Strings
begin
result:=pat([start,findq(ende,'''',#0,s),ende]);
end;
var f1,f2:text;
 ss:string;
 prog:string;
 s1,s2,scomment:string;
 i,p1,apos,alen:integer;
 sl:tstringlist;
 sl2:tstringlist;
 a,b,beginlevel:integer;
 ssf,ssf2,ssf3:string;
begin
// Pascal Kommentare werden an globale Variable skomm zugewiesen
//pcomment:=alt([comm('{','}',skomm),comm('(*','*)',skomm)]);
anchor(1);
ssf:=paramstr(1);
writeln('[',ssf,']');
writeln('lesen von: ',ssf);
assign(f1,ssf);
reset(f1);
sl.count:=0;
sl2.count:=0;
prog:='';
while not eof(f1)do begin
 readln(f1,ss);
 add(sl,ss);
end;
close(f1);
ssf2:=paramstr(2);
```

```
writeln('schreiben nach: ',ssf2);
assign(f2,ssf2);
rewrite(f2);
writeln('Original');
for i:=0 to sl.count-1 do begin
 ss:=sl.a[i];
 writeln(ss);
 // // Kommentar ermitteln, abschneiden und auf Variable scomment
zuweisen
 if mat(ss,0,
[findbq('//',pascalkomm,pascalstring,#0,s1),rem(s2)],apos,alen)then begin
  ss:=s1;scomment:=s2;
 end else scomment:='';
 if debug then writeln('Komm:',ss);
 //begin
 while mat(ss,0,
[findbq(strword('begin'),pascalkomm,pascalstring),'begin',curs(a),rem(s1)
],apos,alen) do begin
  if s1<>''then begin
   add(sl2,copy(ss,1,a));
   ss:=s1;
  end else break;
 end;
 if debug then writeln('begin: ',ss);
 //repeat
 while mat(ss,0,
[findbq(strword('repeat'),pascalkomm,pascalstring),'repeat',curs(a),rem(s
1)],apos,alen) do begin
  if s1<>''then begin
   add(sl2,copy(ss,1,a));
   ss:=s1;
  end else break;
 end;
 if debug then writeln('repeat: ',ss);
 //end
 p1:=0;
 while mat(ss,p1,
[findbq(strword('end'),pascalkomm,pascalstring),curs(b),'end',curs(a)],ap
os,alen) do begin
  if b<>0 then begin
   add(sl2,copy(ss,1,b));
   ss:=copy(ss,b+1,255);
  end else p1:=a;
 end;
 if debug then writeln('end: ',ss);

 //until
 p1:=0;
 while mat(ss,p1,
[findbq(strword('until'),pascalkomm,pascalstring),curs(b),'until',curs(a)
],apos,alen) do begin
  if b<>0 then begin
   add(sl2,copy(ss,1,b));
   ss:=copy(ss,b+1,255);
  end else p1:=a;
```

```
 end;
 if (ss<>'')or(scomment<>'')then
 add(sl2,ss+scomment);
 if debug then begin
  writeln('until: ss: ',ss,' scomment: ',scomment);
  readln;
 end;
end;

writeln;
writeln('Vor Einrücken');
for i:=0 to sl2.count-1 do begin
 writeln(sl2.a[i]);
 writeln(f2,sl2.a[i]);
end;
// Einrücken
sl.count:=0;
beginlevel:=0;
anchor(1);
for i:=0 to sl2.count-1 do begin
 ss:=sl2.a[i];
 ss:=replace(ss,blanks,[''],false,false,true);
 if (ss='')and(beginlevel>0)then continue;
 ss:=dupestring(' ',beginlevel)+ss;
 if
mat(ss,findbq(alt([strword('begin'),strword('repeat')]),pascalkomm,pascal
string))then inc(beginlevel)
 else if
mat(ss,findbq(alt([strword('end'),strword('until')]),pascalkomm,pascalstr
ing))then begin
  dec(beginlevel);
  delete(ss,1,1);
 end;
 add(sl,ss);
end;
Writeln('Nach Einrücken');
for i:=0 to sl.count-1 do begin
 writeln(sl.a[i]);
 writeln(f2,sl.a[i]);
end;
readln;
close(f2);
end.
```

(7.7) Ein arithmetischer Formelrechner

(formel.pas)

Um eine arithmetische Formel mit Klammern auszuwerten müssen ebenfalls nicht reguläre Strukturen erkannt werden.

```
program rechner;
// Taschenrechner mit ( ) + - * / und den Variablen a-z
{$APPTYPE CONSOLE}

uses sysutils,strutils,mysnobol;
```

```
type pdouble=^double;
type pstring=^string;
var vars:array[0..25]of double;

function addop(s:string;var posi:integer;var op:string):boolean;
var a:double;
begin
a:=0.0;
result:=mat(s,posi,any('+-'),op);
if result then posi:=posi+1;
end;

function mulop(s:string;var posi:integer;var op:string):boolean;
begin
result:=mat(s,posi,any('*/'),op);
if result then posi:=posi+1;
end;

function dop(x1,x2:double;s:string):double;
begin
if s='+'then result:=x1+x2
else if s='-'then result:=x1-x2
else if s='*'then result:=x1*x2
else if s='/'then result:=x1/x2
else result:=0;
end;

function statem(s:string;var posi:integer;var x:double):boolean;forward;

function id(s:string;var posi:integer;var x:double;var
s1:string):boolean;
begin
result:=mat(s,posi,any(['a'..'z']),s1);
if result then begin
 x:=vars[ord(s1[1])-ord('a')];
 posi:=posi+1;
end else x:=0;
end;

function number(s:string;var posi:integer;var x:double):boolean;
var s1:string;
begin
result:=mat(s,posi,anyspan(['0'..'9','.']),s1);
if result then begin
 x:=strtofloat(s1);
 posi:=posi+length(s1);
end else x:=0;
end;

function faktor(s:string;var posix:integer;var x:double):boolean;
var s1:string;
 minus:boolean;
 xx:integer;
begin
result:=true;minus:=false;
```

```
while mat(s,posix,stri('-'))do begin
 minus:=not minus;inc(posix);
end;
if id(s,posix,x,s1) then
else if number(s,posix,x)then
else if mat(s,posix,stri('('))then begin
 posix:=posix+1;
 if statem(s,posix,x) and mat(s,posix,stri(')'))
 then begin
  xx:=posix+1;posix:=xx
 end else result:=false;
end else result:=false;
if minus then x:=-x;
end;

function term(s:string;var posi:integer;var x:double):boolean;
var x1:double;
 op:string;
begin
result:=faktor(s,posi,x);
if result then begin
 while mulop(s,posi,op) do begin
  result:=faktor(s,posi,x1);
  if result then x:=dop(x,x1,op);
 end;
end;
end;

function expr(s:string;var posi:integer;var x:double):boolean;
var x1:double;
 op:string;
begin
result:=term(s,posi,x);
if result then begin
 while addop(s,posi,op) do begin
  result:=term(s,posi,x1);
  if result then x:=dop(x,x1,op);
 end;
end;
end;

function statem(s:string;var posi:integer;var x:double):boolean;
var s1:string;
 pos0:integer;
begin
s1:='';result:=false;pos0:=posi;
if id(s,posi,x,s1)then begin
 if mat(s,posi,stri('='))then inc(posi) else posi:=pos0;
end;
if expr(s,posi,x) then begin
 if s1<>'' then vars[ord(s1[1])-ord('a')]:=x;
 result:=true;
end;
end;
```

```
var ss:string;
 x:double;
 posi:integer;
 s:string;
 i:integer;
 a,b:integer;
begin
for i:=0 to 25 do vars[i]:=0.0;
anchor(1);
//trace(20);
write(':');readln(ss);
while ss<>'' do begin
 posi:=0;
 if statem(ss,posi,x) and(posi=length(ss))then begin
  writeln(x:10:2);
 end else begin
  writeln(dupestring(' ',posi),'^');
  writeln('Syntaktischer Fehler');
 end;
 write(':');readln(ss);
end;
end.
```

Statement ist ein Ausdruck mit evtl. Zuweisung
a=3+2*4 Wert wird ausgegeben
expr ist eine Folge von Termen mit Additionsoperatoren
term ist eien Folge von Faktoren mit Multiplikationsoperatoren
faktor ist ein Bezeichner oder eine Zahl oder ein Ausdruck in Klammern