**Referenz der Funktionen**

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 Strings auf apos und alen zurückgegeben. Bei Matching gibt es zwei Varianten:
Nach anchor(true) muss der Pattern im String exakt bereits ab der Position gematcht werden. Nach anchor(false) kann der Pattern irgendwo im String s ab der Position gematcht werden.

function stri(const astr:string):tstri;
Das Pattern matcht den String astr. Bei diesem Pattern wird der Wert des Strings jedoch bei der Erzeugung des Pattern festgelegt. Möchte man den Wert des zu matchenden Strings erst während der Matchingprozedur festlegen, muss man eine Stringvariable benutzen.

function sv(var av:string):tsv;overload;
Es wird der aktuelle Wert der Stringvariablen av eingesetzt. Hiermit kann man sich während des Matching auf bereits erkannte Texte beziehen:
Beispiel:
In einem String sollen alle doppelt vorkommenden Buchstaben nur noch einmal vokommen.

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

Der durch any(buchars,s1) gematchte String 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.

function cat(aarr:array of const):tcat;overload;
Der Parameter von cat ist eine Liste von Strings 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 Methode genommen. D.h. bei matchnext können s1 und s2 andere Werte enthalten.

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

type tstringfunction=function(s:string):string;
function sf(af:tstringfunction;var s:string):tsf;
Auf die Stringvariable s wird die Funktion af angewandt und das Ergebnis genommen.
af kann eine beliebige vom Benutzer geschriebene Funktion sein. Diese Funktion muss aber die Form: function(s:string):string haben.

function iv(var av:integer):tiv;overload;
Ergebnis ist der aktuelle Wert der Variablen av als String.

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

Ergebnis ist die Summe der Parameter als String

function minus(av:tiv):tminus;
Ergebnis ist der negative Wert von av

function bal(const klastr:string):tbal;overload;
Mit bal(',[,]') werden Strings erkannt, die jeweils eine balancierte Zahl von öffnenden und schließenden Klammern [ und ] enthalten. 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.

function bal(const klastr:string;var s:string):tbal;overload;
Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

function quote(const quotestr:string):tbal;overload;
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 beendet, 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.
function quote(const quotestr:string;var s:string):tbal;overload;
Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

function balquote(const klastr,quotestr:string):tbal;overload;
Die Funktion erkennt balancierte Strings mit eingeschlossenen Quotes.

function balquote(const klastr,quotestr:string;var s:string):tbal;overload;
Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

function bal(const klastr,quotestr:string;aslash:char):tbal;overload;
Hier wird zusätzlich ein Fluchtzeichen angegeben. D.h. Zeichen hinter dem Fluchtzeichen werden nicht als Klammer- oder Quotezeichen erkannt.

function bal(const klastr,quotestr:string;aslash:char;var s:string):tbal;overload;
Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

function bal(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char):tbal;
overload;
Die Parameter können auch in Form eines Array angegeben werden.

function bal(aklaauf,aklazu,aquoteauf,aquotezu:array of const;aslash:char;var
s:string):tbal; overload;
Der Aufruf kann auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

Mit bal(',[,]') werden Strings erkannt, die jeweils eine balancierte Zahl von öffnenden und schließenden Klammern [ und ] enthalten. 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. 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 öffnendem 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. Siehe Beispiele weiter unten.

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

function any(const astr:string):tany;overload;
any erkennt jedes Zeichen aus dem String astr

function any(const achars:tcharset):tany;overload;
Hier wird jedes Zeichen aus der Zeichenmenge achars erkannt.

function any(const astr:string;var s:string):tany;overload;
function any(const achars:tcharset;var s:string):tany;overload;
Die Aufrufe können auch optional einen var-Parameter enthalten, auf dem der gematchte String abgelegt wird.

function any(avv:tsv):tany;overload;
function any(avv:tsv;var s:string):tany;overload;
Als Parameter kann auch eine Stringvariable genommen werde, die erst während des Matching ihren Wert erhält (durch sv).

Beispiele:
any('AEIOU') matcht jeden Vokal
any('0'..'9') matcht jede Ziffer

Der Parameter von tany kann auch zur Laufzeit erst festgelegt werden, durch eine Instanz von tsv als Parameter.

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;

Der Gegensatz von any ist notany. Hier wird jedes Zeichen gematcht, das nicht aus einer vorgegebenen Zeichenmenge stammt.

Mit der Funktion span werden Strings 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 Zeichenmange, sonst liefert anyspan fail als Ergebnis. Bei span(const astr:string) wird die Zeichenmenge aus den Zeichen des Strings gebildet. Mit span(avv:tsv) kann die dieser String auch beim Matchen gebildet werden.

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 matcht den String bis zu einem Zeichen aus dem ersten Parameter.
upto ist immer erfolgreich, da auch der leere String gematcht wird.

function len(ai:integer):tlen;overload;
function len(ai:integer;var s:string):tlen;overload;
Möchte man einen beliebigen String fester Länge matchen kann man die Funktion len benutzen.
Das Matching ist nicht erfolgreich, wenn der Reststring kürzer als die verlangte Länge ist.
Bei len gibt es keine Alternative.
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;
Bei der Funktion arb wird jeder String, ab einer vorgegeben Länge gematcht. Bei arb(0) entsprechend jeder String.
Beim ersten Durchlauf wird ein String mit minimaler Länge gematcht. Bei arb bzw arb(0) der leere String. Bei jedem Rematch wird dann die Länge jeweils um eins erhöht. Der Match ist nicht erfolgreich, wenn die Restlänge nicht mehr ausreicht.

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;

Möchte man den Teilstring von der aktuellen Position bis zu einer gegebenen Position im String matchen kann man die Funktion tab bzw rtab benutzen. Bei tab wird die Position vom Anfang gerechnet, bei rtab 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 tab(av:tiv) kann die Position auch beim Matchen erst festgelegt werden, z.B. eine feste Distanz zu einer vorher mit tcurs (s.u.) ermittelten Cursorposition.
Durch einen zusätzlichen Parameter kann man bei den meisten Pattern auch angeben, auf welcher Stringvariablen der gematchte Teilstring abgespeichert werden soll.

function curs(var apos:integer):tcurs;
Möchte man jedoch die aktuelle Cursorposition abspeichern kann man tcurs benutzen. curs(i) matcht immer erfolgreich den leeren String und liefert die aktuelle Cursorposition auf der Variablen i.

function fpos(aposi:integer):tfpos;overload;
function fpos(av:tiv):tfpos;overload;
Bei einem Aufruf von fpos(i) wird ebenfalls der leere String gematcht, jedoch nur, wenn die aktuelle Cursorposition i ist.

function rpos(aposi:integer):trpos;overload;
function rpos(av:tiv):trpos;overload;
Bei rpos(i) wird die Cursorposition entsprechend vom Ende gerechnet.

function pat(apat:array of const):tpat;overload;
function pat(apat:array of const;var s:string):tpat;overload;
Mit pat wird ein zusammengesetztes Pattern definiert.
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
- Pattern

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:
pat([any(bu),span(bu+zi)])
matcht Bezeichner

function pat(astr:string):tstri;overload;
Hier wird der String astr gematcht

function pat(ai:integer):tarb;overload;

Hier wird ein beliebiger String der Länge ai gematcht. Bei ai=0 wird jeder String gematcht.

function pat(ai:integer;var s:string):tarb;overload;
Der gematchte String wird in s abgespeichert.

Eine besondere Rolle spielt als Parameter von pat eine Instanz der Klasse tca.
function ca(var av:string):tca;
Bei ca(s) erhält bei erfolgreichem Match des gesamten Patterns die Variable s den vom vorhergehenden Teilpattern gematchten String.
Diese Funktion kann jedoch fast immer ersetzt werden durch dem optionalen Parameter bei den Pattern Aufrufen, der eine Stringvariable enthält, auf die der gematchte String gespeichert wird.

Mit der Funktion alt 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;

Es wird zunächst die erste Alternative der ersten Pattern gematcht. Sind beim ersten Pattern alle Alternativen erschöpft, wird die erste Alterantive des zweiten Patterns gematcht. Der Match ist nicht erfolgreich, wenn alle Alternativen des letzten Patterns ausgeschöpft sind.

Um reguläre Strukturen zu erkennen, muss man auch Wiederholungen von Pattern mit vorgegebener Anzahl erkennen. Dazu dient die Funktion rep.
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 Alterantive.
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

```
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.

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.

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.

function look(alpat:tpattern):tlook;overload;
function look(s:string;alpat:tpattern):tlook;overload;

Jeder String hinter dem alpat folgt. s String bis alpat
function look(apat,alpat:tpattern;aanchor:boolean):tlook;overload;
function look(apat:tpattern;var s:string;alpat:tpattern;aanchor:boolean):tlook;
overload;
Bei aanchor=false String gefolgt von apat und alpat. Bei aanchor=true apat gefolgt von alpat. s enthält String und apat-Match.
function look(var sbis:string;apat:tpattern;var s:string;alpat:tpattern):tlook;overload;
Anchor=false und sbis enthält den String vor apat.

Ebenfalls nach einem Pattern sucht die Funktion find. Zusätzlich können für den String bis zum gesuchten Pattern Klammer- und Quotebedingungen angegeben werde, wie bei tbal.
function find(apat:tpattern):tfind;overload;
Sucht das Pattern apat
function find(apat:tpattern;var abis,s:string):tfind;overload;
Der String bis zum Pattern apat wird bei sbis abgelegt, der von apat gematchte bei s.
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 abis,s:string):tfind;overload;
Der String bis zum Pattern apat muss die Klammer- und Quotebedingungen erfüllen.
function findb(apat:tpattern;const klastr:string):tfind;overload;
function findb(apat:tpattern;const klastr:string;var abis,s:string):tfind;overload;
Der String bis zum Pattern apat muss die Klammerbedingungen erfüllen.
function findbq(apat:tpattern;const klastr,quotestr:string):tfind;overload;
function findbq(apat:tpattern;const klastr,quotestr:string;var abis,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
abis,s:string):tfind; overload;
Der String bis zum Pattern apat muss die Klammer- und Quotebedingungen erfüllen.
function findq(apat:tpattern;const quotestr:string):tfind;overload;
function findq(apat:tpattern;const quotestr:string;var abis,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 abis,s:string):tfind;
overload;
Der String bis zum Pattern apat muss die Quotebedingungen erfüllen.
function find(const astr:string):tfind;overload;
function find(const astr:string;var abis,s:string):tfind;overload;
Es wird der String astr gesucht
function findb(const astr:string;const klastr:string):tfind;overload;
function findb(const astr:string;const klastr:string;var abis,s:string):tfind;overload;
Der String bis zum String astr muss die Klammer- und Quotebedingungen erfüllen.
function findbq(const astr:string;const klastr,quotestr:string):tfind;overload;
function findbq(const astr:string;const klastr,quotestr:string;var abis,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
abis,s:string):tfind; overload;
Der String bis zum String astr muss die Klammer- und Quotebedingungen erfüllen.
function findq(const astr:string;const quotestr:string):tfind;overload;

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

Um den Patternmatching-Algorithmus abzubrechen bzw. weitere Alternativen zu suchen gibt es einige spezielle Pattern.
Kommt der Algorithmus an eine Instanz von tabort bricht er sofort ohne Erfolg ab.
function abort:tabort;

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;

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;

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).

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

Mit der Funktion repl 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.

**Funktionen zum Patternmatching**

function mat(var s:string;ab:integer;const pat:tpattern):boolean;overload

Test: s matcht 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 Strings 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 Strings 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 Strings setzen. Bei anchorglb=true muss genau am Anfang gematcht werden

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:tsv;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 beim Matchen 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 Teilstrings 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:tsv; 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.

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; ab:integer=0):string; overload;
function match(s:string;arr:array of const;rep:boolean=false;rek:boolean=false; ancho:boolean=false;ab:integer=0):string;overload;

Möchte man alle Vorkommen eines Pattern in einem String ermitteln, benutzt man die Funktion matchall.
function matchall(var s:string;ab:integer;pat:tpattern;var apos:tintarray;var sl:tstringlist): 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 der Stringliste sl und die Position in der Integerliste aposl;
function matchall(var s:string;ab:integer;pat:array of const;var apos:tintarray;var sl:tstringlist):integer;overload;
Analog mit der Patternfolge pat

function matchall(var s:string;ab:integer;pat:tpattern;var sl:tstringlist):integer; overload;
In s werden alle Teilstrings, die pat matchen ermittelt; sl liefert die gematchten Strings

Beispiel: matchall(ss,0,name,sl) liefert alle Namen in der Stringliste ss
function matchall(var s:string;ab:integer;pat:array of const;var sl:tstringlist):integer; overload;
Analog mit der Paternfolge pat

Will man alle Sätze einer Datei bearbeiten, kann man die Prozedur substitutefile benutzen.
procedure substitutefile(const quelle,ziel:string;arr: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 match(s,arr,rep,rek,ancho,ab) aufgerufen. Der Funktionswert bildet die Zeile in der Datei ziel.

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.

**Vordefinierte Mengen und Pattern**
const buchars:tcharset=['A'..'Z','a'..'z','Ä','Ö','Ü','ä','ö','ü','ß'];
Alle Buchstaben
const zichars: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(buchars),span(namechars)]);
Alle Bezeichner
var blanks:tpattern=span(' ');
Längste Folge von Blanks