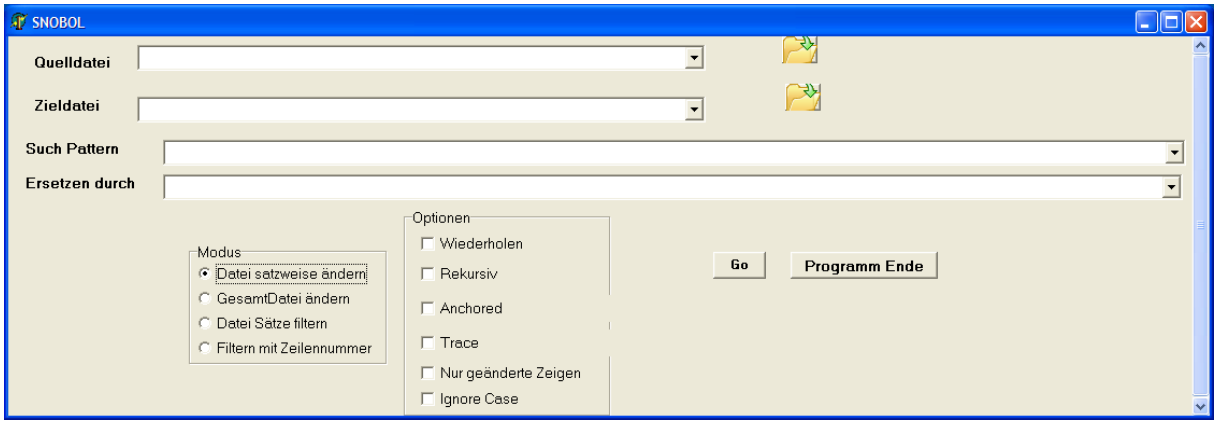
Die SNOBOL IDE



Die SnobolIDE erlaubt es für Textersetzungen, die SNOBOL Funktionen zu benutzen, ohne zuerst ein entsprechendes Programm zu schreiben. Die Quelldatei wird entsprechend dem angegeben Pattern modifiziert und auf die Zieldatei geschrieben. Ist das Feld für die Zieldatei leer, wird auf die Standardsausgabe geschrieben.

Es gibt 3 Modi:

1. Datei Satzweise ändern.
   Hier wird jeder Satz für sich mit den angegeben Pattern bearbeitet
2. Gesamtdatei ändern
   Hier werden die Patterns auf die gesamte Datei angewandt. Die einzelnen Sätze sind durch CR/LF getrennt.
3. Datei Sätze filtern
   Ist das Suchpattern erfolgreich, wird der Satz ausgegeben

In den Patterns können die Stringvariablen s1 bis s9 benutzt werden.

Im **Modus 1** wird auf jeden Satz **`sin`** angewandt:

```
sout:=replace(sin,[<Such Pattern>],[<NeuesPatterns>],rep,rek)
```

Die Pattern können mehrere Pattern sein, die durch Komme getrennt werden.
Im Ersetzen Pattern kann man sich auf die String Variablen s1 bis s9 beziehen. Dazu wird jeweils das Pattern z.B. sv(s1) eingefügt. An dieser Stelle wird der Wert von s1 eingefügt. Die Variable s1 wird durch einen Zuweisungsparameter im Such Pattern gesetzt. Die Variablen s1 bis s9 können so ihre Werte durch Pattern erhalten, die den gematchten String an eine Variable zuweisen.

Beispiel:
Such Pattern: any(letters,s1),span(namechars,s2)
Ersetzen Ausdruck: sf(uppercase,s1),sf(lowercase,s2)
Das Such Pattern findet alle Namen. Der erste Buchstaben wird der Variablen s1 zugewiesen, der Rest der Variablen s2.
Für s1 wird die Funktion uppercase ausgeführt, für s2 die Funktion lowercase. Man kann die Funktionen uppercase und lowercase nicht direkt aufrufen, da für s1 bzw. s2 nicht der wert bei Aufruf genommen werden kann, sondern erst der nach

Matching. Mit sf wird nach dem Matching die Funktion aufgerufen. Die aufgerufene Funktion muss jeweils vonder Form: **function(s:string):string** sein.
Die IDE setzt die Pattern jeweils in eckige Klammern, wie es für das Snobol Programm notwendig ist..
Wenn wiederholen nicht gesetzt wird, wird jeweils nur der erste Name geändert, sonst alle.

Im **Modus 2** wird wie im Modus 1 verfahren, jedoch wird nicht jeder Satz einzeln bearbeitet, sondern die Datei als ganzes.

Beispiel:
Such Pattern: **plus(#13#10)**
Ersetzen Ausdruck: **#13#10**

Das Pattern plus(#13#10) findet alle ein oder mehrmals auftretenden CR/LF. Diese werden ersetzt durch jeweils einen CR/LF

Im **Modus 3** wird das Suchpattern auf jeden Satz angewandt. Wenn das Suchpattern matcht, wird der Satz ausgegeben.
Beispiel:

Such Pattern: **alt(['procedure','function'])**

Es werden alle Sätze mit Prozedur- oder Funktionsdeklarationen ausgegeben. Möchte man sicherstellen, das procedure oder function nicht Teil eines anderen Strings ist gibt man an:
Such Pattern: **wordstart,alt(['procedure','function']),wordend**

Bestimmen der Namen der Zieldateien

Im Opendialog können mehrere Dateien ausgewählt werden. Die Namen der Zieldateien werden wie folgt bestimmt:
mit *.pa1 z.B. erhalten die Zieldateien jeweils die Erweiterung .pa1
mit save\ landen die Zieldateien mit dem Originalnamen im Verzeichnis save.
Beginnt das Zielverzeichnis nicht mit \ oder c:\ wird das Zielverzeichnis relativ zum Quellverzeichnis angenommen sonst als absoluter Verzeichnisname.

Wird keine Zieldatei angegeben, wird alles auf die Konsole ausgegeben.
Wird + als Zieldatei angegeben wird die Quelldatei jeweils geändert.
Zweckmäßigerweise sichert man sie vorher wo anders.

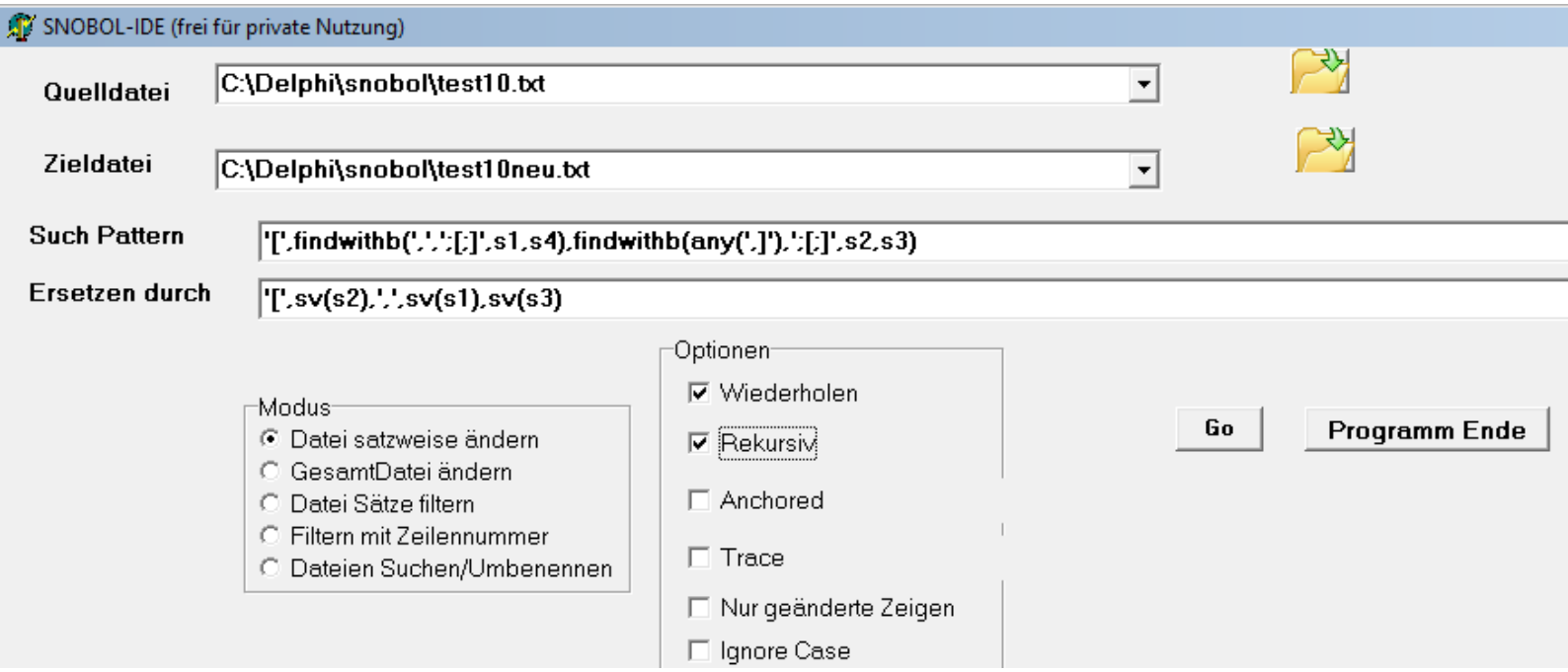
Optionen
- Wiederholen Es werden alle Matchings geändert
- Rekursiv Auf den gematchten String, wird die Ersetzen Funktion wieder angewandt, bis kein Matching mehr möglich ist.
- Anchored Das Matching muss beim 1. Zeichen beginnen
- Trace Das Matching wird protokolliert
- Nur geänderte zeigen Es werden nur die Zeilen ausgegeben, die geändert wurden

- Ignore Case Groß- Kleinschreibung wird ignoriert

Mit **Go** wird das Matching gestartet.

Die IDE erzeugt aus der Eingabe eine Pascaldatei, die im Verzeichnis
users\<user>\appdata\roaming
abgespeichert wird. Auf dieses Verzeichnis hat auch ein Benutzer mit eingeschränkten Rechten Zugriff. Dann wird Pscript mit dieser Datei aufgerufen. Beim Beenden wartet Pscript auf eine leere Eingabe. Dabei können evtl. Fehlermeldungen gesehen werden.

Beispiel:



Hier wird in einem Array mit 2 oder mehr Dimensionen jeweils die 1. und die 2. Komponente getauscht.
Der Suchpattern findet [. Dann sucht er das nächste Komma auf der gleichen Ebene, Der Text bis dahin wird auf s1 gespeichert.
Der nächste Pattern findet das nächste , oder] auf der gleichen Ebene. Der Text bis dahin wird auf s2 gespeichert. Auf s3 wird das gefundene Zeichen , oder] gespeichert.
Das gefundene wird ersetzt durch [, den Wert von s2 (die 2. Komponente),ein Komma und dann die erste Komponente (s1). Anschließend das gefunden Zeichen,] bei einem 2 dimensionalen Array bzw. Komma bei 3 und mehr Dimensionen.
Durch findwithb (balanced) und Angabe der Klammersymbole [und] wird erreicht das Pattern nur auf der gleichen Ebene gefunden wird.
Die Klammersymbole werden durch einen String definiert.
`;[;]`
Das erste Zeichen ist das Trennzeichen, dann folgen die öffnende und die schließende Zeichenfolge. Diese darf auch mehr als ein Zeichen enthalten.
z.B ;(*;*) für Pascal Kommentare.

Das obige Beispiel würde so nicht funktionieren, wenn in dem Ausdruck ein Kommentar vorkommt, der [oder] enthält. Daher kann man im Klammerausdruck auch mehrere Klammerpaare angeben.
Für Pascal wäre dies dann

```
,[,],{,},(*,*)
```