RUHR-UNIVERSITÄT BOCHUM

ARBEITSBERICHTE

des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7602 BO.E6.03

# BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL60

2. erweiterte und geänderte Version

von

Manfred Rosendahl

März 1976

4630 Bochum, Universitätsstr. 150, Geb. NA

INHALTSVERZEICHNIS:

## 0. Einleitung

Die Algol 60 Implementierung des TR 440 [1] ist eine sehr weitegehende und vollständige Implementierung der Sprache Algol 60 [2]. Die Sprache Algol 60 enthält jedoch keine Anweisungen zur Stringbearbeitung. Strings sind zwar als Daten im Sprachumfang enthalten, können jedoch nur ein- und ausgegeben und als formale Parameter übergeben werden. Lediglich für sehr beschränkte Vergleiche steht die Prozedur EQUIV zur Verfügung.

Als eine zur Zeichenkettenverarbeitung sehr geeignete Sprache gilt SNOBOL. Gebräuchlich sind die beiden Versionen SNOBOL 3 [3] und SNOBOL 4 [4].

Im Rahmen des Programmiersystems BOGOL-STRING werden die meisten Möglichkeiten von SNOBOL 4 in Algol-Programmen ansprechbar. Zusätzlich ist eine syntaxgesteuerte Textverarbeitung, ähnlich in CDL [8,9] möglich.

BOGOL-STRING ist ein Teil des Systems BOGOL (BOchumer-alGOL) [1)], das zur Erweiterung der Algol Sprachmöglichkeiten gedacht ist.

## 1. Überblick über das System BOGOL-STRING

### 1.1. Aufbau des Systems

Das System BOGOL-STRING ist als Unterprogrammpaket aufgebaut. Eine Reihe von Prozeduren erlauben die Stringsverarbeitungsmöglichkeiten von SNOBOL 4 [4] insbesondere zur kontextabhängigen Stringuntersuchung, und von CDL [8,9] zur syntaxgesteuerten Stringerkennung und Generierung.

Die Prozeduren MAT und ASS simulieren ein SNOBOL-Statment.

MAT (A, $B_1$,...,$B_n$) ;

Der String A wird auf die Bedingungen $B_1$,...,$B_n$ getestet.

ASS (A, $B_1$,...,$B_n$, C);

Der String A wird auf die Bedingungen $B_1$,...,$B_n$ getestet und der diese Bedingungen erfüllende Teilstring wird durch den String C ersetzt.

Mit Hilfe der Prozeduren RU, RS, STAR und GO kann eine Syntax ähnlich wie in CDL durch eine Affix-Grammatik [12] beschrieben werden. Diese Syntax kann ebenfalls als Testbedingung in MAT oder ASS angegeben werden.

---

1) Weitere Teile des BOGOL-Systems:
(1) BOGOL-TAS, zur systemnahen Programmierung in Algol (bereits implementiert) [5]. Neubearbeitung in Vorbereitung.

In Vorbereitung:
(2) BOGOL-DATA [6] . Erlaubt es in Algol 60 strukturierte Datenstrukturen (modes) wie in ALGOL68 aufzubauen und zu verarbeiten.
(3) BOGOL-FORMAC, zur Formelmanipulation und zum symbolischen Rechnen.

## 1.2. Deklaration und Handhabung

Die Darstellung der Strings ist die gewöhnliche Algol-TR 440 Stringdarstellung, also:

| | |
|---|---|
| 1. Ganzwort: | Länge des Strings mit TK1, |
| 2. und folgende Ganzworte: | Stringinhalt mit 6 Zeichen/Wort. |
| BEISPIEL: | UNI-BOCHUM |

| 1 | 10 | 3 | UNI-BO | 3 | CHUM |
|---|---|---|---|---|---|

Ein solcher String wird in aufeinanderfolgenden Zellen i. A. einem Array abgespeichert.

Ein so abgespeicherter String kann wie ein gewöhnlicher Algol-string behandelt werden, also als Parameter auftreten oder ausgegeben werden.

Er entspricht jedoch nicht dem A-Format der Knuth-Prozedur INPUT bzw. OUTPUT.

Die BOGOL-STRING Prozeduren werden wie gewöhnliche Algol code-Prozeduren in der Form:

⟨type⟩ procedure NAME(X); code;

deklariert. Wegen der Typenkennung ist jedoch folgendes zu beachten:

Wird einer integer-Größe eine real-Größe zugewiesen, so erfolgt eine Rundung. Dies läuft jedoch für TK ≠ 0 auf einen Alarm. Daher werden Strings zweckmäßig nur auf real Variablen abgespeichert.

Die Prozeduren des BOGOL-Systems liegen als Montageobjekte in der Bibliothek BOGOL auf der LFD (langfristige Datenhaltung) vor.

Nach dem Kommando □ BIBANM., BOGOL sind sie wie gewöhnliche Algol-Prozeduren der &OEFDB (öffentliche Datenbasis) zu benutzen.

Zur Vermeidung des lästigen Deklarierens der code- Prozeduren ist ein Precompiler implementiert, der in einem Algol-Programm fehlende Deklarationen einsetzt. (Kommando PREKOM [ 7 ] ).

## 1.3. Bei den String- und "Match"-Prozeduren auftretende Parametertypen

Bei den Stringprozeduren können mehr als die gewöhnlichen Algol 60 Parametertypen vorkommen.

In den Prozedurbeschreibungen werden für die erlaubten Typen der aktuellen Parameter folgende Bezeichnungen verwandt:

| | |
|---|---|
| stringvar | Feld(name) oder Variable, aber der ein String abgelegt werden kann. |
| string | String, Funktionsaufruf der ein String liefert<br>Feld das String enthält, Variable ab der ein String beginnt |
| array | Feldname |
| ref | Referenz (Adresse) einer Variablen. Dies kann erreicht werden durch Angabe der Variablen, wenn ihr Inhalt eine Adresse ist oder durch einen Ausdruck, der die Adresse angibt, z. B. durch Prozedur REF(BOGOL-TAS [5] ). |
| integer | Algol-Variable oder Größe mit integer-Wert. |
| real | Algol-Variable oder Größe mit beliebigem Wert. |
| pattern | siehe Prozedur MAT |
| var | Variable, keine Konstante oder Ausdruck. Kann auch ein Prozeduraufruf sein, der Wert und Adresse liefert, z. B. VAL. |

Diese Bezeichnungen dienen nur zur Beschreibung. Sie haben keine Beziehung zur aktuellen Deklaration im Programm. Zur Deklaration siehe 1.2.

## 2. "Pattern-matching"

Diese Prozeduren zur Prüfung des Auftretens bestimmter Teilstrings in einem String (das "pattern-matching") sind die Hauptprozeduren zur String-manipulation.

Die Prüfung kann in den Prozeduren:

MAT (nur Prüfung),
ASS (Prüfung mit Zuweisung (2.2.)) und
PAT (Prüfung einer zusammengesetzten Bedingung (2.4.2) erfolgen.

### 2.1. "Matching" Prozedur MAT

Die "pattern-matching"-Prozedur MAT hat folgendes Format:

MAT (A, $B_1$, ..., $B_n$);
string oder real A;
pattern $B_1$, ..., $B_n$.

Bei A wird der zu untersuchende String angegeben oder von wo der zu untersuchende String eingelesen wird.

Dabei gibt es folgende Möglichkeiten:

| | |
|---|---|
| A = 0 : | Lesen von Konsole (Gespräch) bzw. Fremdstring (Abschnitt) und zwar jeweils eine Zeile. |
| 0 < A < 100 : | Der zu testende String wird von der Gerätenummer A eingelesen. |
| 100 < A : | Es sei A = 100 · S + G.<br>Der zu testende String ist der Satz Nr. S von der Random-Datei mit Ger.-Nr. G. |
| A < 0 : | Gelesen wird wie bei /A/ jedoch wird der gelesene String an den zuletzt getesteten angehängt. |
| A = 0.5 : | Lesen von Konsole/Fremdstring mit Anhängen. |

Das Anhängen des gelesenen Strings an den bisher getesteten erlaubt es zusammen mit IPUSH (Linksshif 2.7.3) beliebig lange Strings, auch über mehrere Zeilen, zu testen.

Die Parameter $B_1$ bis $B_n$ sind die Testbedingungen (pattern).

Dies Parameter spezifizieren gewisse Teilstrings oder Bedingungen, die fortlaufend im String A vorkommen bzw. erfüllt sein müssen, damit der "pattern-match" erfolgreich ist. Dieser Erfolg kann nachher abgefragt werden (SUCC, FALL). Dieses "matching" testet ein "scanner"- Algorithmus, der die einzelnen Bedingungen überprüft. In einigen Fällen können mehrere evtl. verschieden lange Teilstrings die Bedingung erfüllen. Dann wird zunächst die erste Alternative (bei ALT) bzw. der kürzeste mögliche Teilstring (evtl. auch der Nullstring) genommen. Kann nachher eine spätere Bedingung nicht erfüllt werden, so wird wieder zurückgegangen und eine andere Alternative oder ein längerer Teilstring genommen.

Der "pattern-matching" Algorithmus verläuft genauso wie in SNOBOL nach folgenden Regeln:

Zunächst im Mode "unanchored".

(1) Es wird versucht, die erste Bedingung (z. B. ein String) vom 1. Zeichen des zu untersuchenden Strings A an zu matchen. Geht dies nicht, dann vom 2. Zeichen an und so weiter.

(2) Es werden nach rechts fortlaufend alle folgenden Bedingungen durch aufeinanderfolgende Teilstrings von A zu "matchen" versucht.

(3) Kann eine Bedingung nicht erfüllt werden, so wird versucht, die vorhergehende Bedingung durch eine andere Alternative (bei Prozedur ALT) oder einen längeren Teilstring zu "matchen".

(4) Hat die vorhergehende Bedingung keine freie Alternative mehr bzw. kann die Länge des "matchenden" Teilstrings nicht vergrößert werden, so wird weiter nach links gegangen und die davor liegende Bedingung untersucht usw.

Der "matching"-Algorithmus läuft also hin und her bis entweder

a) die letzte Bedingung "gematcht" wurde, dann ist der "pattern match" erfolgreich (SUCC) oder

b) die erste Bedingung mangels freier Alternativen oder nicht mehr zu verlängerndem Teilstring nicht "gematcht" werden kann. Dann war der "pattern match" ohne Erfolg (FAIL).

Anschließend liefern die boolschen Prozeduren SUCC und FAIL die entsprechenden Werte.

Im Mode " anchored ", einstellbar durch ANCHOR(1), muß die erste Bedingung durch einen Teilstring, der mit dem ersten Zeichen des zu untersuchenden Strings beginnt, "gematcht" werden.

Es ist der Mode "unanchored" eingestellt.

Die einzelnen Pattern $B_i$ bis $B_n$ können sein:

| | |
|---|---|
| integer n $\geq$ 1 | Dies "matcht" jeden Teilstring der Länge n |
| 0 | Diese Bedingung "matcht" jeden beliebigen Teilstring ( ARB in SNOBOL4 ). |
| string | Diese Bedingung "matcht" den angegebenen String als Teilstring. |

Aufruf einer "matching"-Prozedur (m-procedure).

Dies sind:

ALT, PAT, (2.4)

ANY, NOTANY (2.5.1)

SPAN, BREAK (2.5.2)

TAB, RTAB, REM (2.5.3)

POS, RPOS (2.5.4)

ARB, ARBNO, LEN, BAL, UNTIL (2.5.5 - 2.5.8)

FAILURE, SUCEED (2.5.9)

FENCE, NOT, DO, IP, OP (2.5.10 - 13)

RU, RS, STAR (2.6) .

Diese Prozeduren "matchen" jeweils einen durch sie definierten Teilstring. Ferner können sogenannte Zuweisungsprozeduren angegeben werden.

Dies sind:

SV, SVOP, OSV, CURS .

Hierbei wird jeweils der Nullstring "gematcht" und der die letzte Bedingung matchende Teilstring wird zugewiesen. Bei CURS dagegen die Cursor-Position.

Schließlich können noch Prozeduraufrufe angegeben werden, die den Cursor neu positionieren bzw. den zu "matchenden" String verlängern bzw. shiften.

Dies sind:

BACK, INPU, ISPUSH (2.7).

Alle diese Prozeduren werden im Programm einfach als real procedure deklariert.

BEISPIEL: String A sei: 'LAND UND WASSER'

```
MAT (A,'('LAND')',O,'('WASSER')');
```

O "matcht" den Teilstring: ' UND '

Der Aufruf `MAT (A,'('UND')')`

im Mode "unanchored" ist äquivalent zum Aufruf:

`MAT (A,O,'('UND')')` im Mode "anchored".

## 2.2. Ersetzungs-Prozedur: ASS

Die Ersetzungsprozedur ASS dient dazu, in einem String einen Teilstring zu ändern. Dies kann in Abhängigkeit eines "matching"-Tests gemacht werden. Der String darf jedoch nicht ein expliziter String sein, da dieser schreibgeschützt ist, also nicht verändert werden kann.

Die Ersetzungsprozedur ASS kann vor der Ersetzung wie die Prozedur MAT einen "matching"-Test durchführen. Die Ersetzung hängt dann von dem Testergebnis ab. Der Aufruf hat die Form:

ASS(A, $B_1$,..., $B_n$, C);
string A, C;
pattern $B_1$,..., $B_n$.

A darf jedoch kein expliziter String sein.

Zunächst wird wie beim Prozeduraufruf

MAT (A, $B_1$,..., $B_n$)

ein Matching-Test durchgeführt. Ist dieser nicht erfolgreich, so bewirkt die Prozedur ASS keine Ersetzung. Ist der Test jedoch efolgreich und die Elemente $a_i$, $a_{i+1}$,..., $a_j$ "matchen" die Bedingungen $B_1$,..., $B_n$, so erhält A die Gestalt: $a_1$,...,$a_{i-1}$,$c_1$,...,$c_k$,$a_{j+1}$,...,$a_n$. Also $a_i$,...,$a_j$ werden durch C = $c_1$...$c_k$ ersetzt. Die Möglichkeiten für die Parameter sind die gleichen wie bei den Prozeduren MAT.

BEISPIEL: String A = 'LAND UND WASSER'
ASS (A, '('UND')','('ODER')'); liefert:
A = 'LAND ODER WASSER'

## 2.3. Wertzuweisung während des "matching"-Algorithmus:

### 2.3.1. Wertzuweisung an Variablen SV

Tritt in den Spezifikationsparametern $B_1$ bis $B_n$ des "matching"-Algorithmus eine Parameterfolge:

...,$B_i$, SV(D),...

auf, so wird der Teilstring, der die Spezifikation $B_i$ "gematched" hat, auf die Stringvariable D zugewiesen. D kann also array oder var (stringvar) sein.

BEISPIEL: array A, D [1:10];
MAT (A, '('E')',0,SV(D),'('E')');

Der erste Teilstring, der zwischen zwei 'E' steht, wird auf D zugewiesen, jedoch nur, falls das 2. 'E' gefunden wird.

### 2.3.2. Wertzuweisung an die Ausgabe SVOP

m-procedure SVOP;

SVOP 'matcht' den Nullstring und entspricht, falls B ein genügend großes Feld ist, den Aufrufen SV(B), OP(B).

Das heißt, der zuletzt 'gematchte' Teilstring wird an den Ausgabestring angefügt (Ausgabestring siehe OP 2.5.13).

2.3.3. Wertzuweisung der letzten Ausgabe OSV

```
m-procedure OSV(A);
stringvar A;
```

OSV matcht ebenfalls den Nullstring. Auf A wird der im vorigen Parameter an den Ausgabestring (2.5.13) angefügte Teilstring zugewiesen.

BEISPIEL: Sei S ein String, so enthält das Feld B nach

```
MAT(S, PAT(POS(0), 0, SV(B), RPOS(0)),
       PAT(OP('('[')'), OP(B), OP('('⌋')')), OSV(B)));
```

den im String S in [ .. ] gesetzt.

2.3.4. CURS

```
m-procedure CURS(A);
var A;
```

Die Prozedur CURS "matcht" immer den Nullstring, hat also für den "matching"-Algorithmus keine Bedeutung. Es wird jedoch die aktuelle Cursorposition auf der Variablen A abgelegt. Die Cursorposition ist die Nummer desjenigen Zeichens, das zuletzt "gematcht" wurde. Am Anfang ist sie also 0.

BEIPIEL:

```
array A [1:10] ; real B;
MAT(A, POS(0), SPAN('(' ')'), CURS(B));
```

Die Anzahl der führenden Blanks wird auf B abgelegt.

## 2.4. Zusammengesetzte Pattern

### 2.4.1. Alternativen ALT

```
m-procedure ALT (A1,A2,...,An);
pattern A1,...,An;
```

Sollen für eine "matching"-Bedingung mehrere Alternativen bestehen, so kann dies durch die Prozedur ALT angegeben werden. Die verschiedenen Alternativen sind die Parameter der Prozedur ALT. Sie werden der Reihe nach getestet. Ist eine erfüllt, wird im Algorithmus weitergegangen. Kann später der "matching"-Algorithmus eine Bedingung nicht erfüllen, so wird zurückgegangen, und es werden die übrigen folgenden Alternativen untersucht. Die Parameter von ALT können alle Typenwerte sein, die auch sonst für die Spezifikationen $B_i$ erlaubt sind.

BEISPIEL:

```
array A [1:10] ;
MAT(A, ALT('('ALT')', '('NEU')'), ALT('('DORF')', '('STADT')'));
```

Erfolgreich "gematcht" werden die Strings:

ALTDORF
NEUDORF
ALTSTADT
NEUSTADT

### 2.4.2. Sequentielle Pattern PAT

Ein Pattern ist eine Kombination von alternativ oder hintereinander zu erfüllenden Match-Bedingungen. Dies wird definiert durch die Prozedur PAT.

m-procedure PAT ($B_1$,...,$B_n$);
pattern $B_1$,...,$B_n$ ;

Die Prozedur PAT ist erfolgreich, wenn vor der aktuellen Cursorposition an die Bedingungen $B_1$ bis $B_n$ hintereinander erfüllt werden können. Falls eine Bedingung nicht erfüllt werden kann, wird versucht bei einer vorigen Bedingung, falls vorhanden, eine andere Alternative zu testen. Der Algorithmus ist so der gleiche, wie bei MAT (2.1) beschrieben.

BEISPIEL: Im Mode "anchored" ist der Aufruf

MAT (A, B, C, D) äquivalent zu

MAT (A, PAT (B, C, D));

Dagegen ist: ANCHOR(O); MAT (A, B, C, D) äquivalent zu:

ANCHOR(1); MAT (A, PAT(O, B, C, D));

## 2.5. Primitive "Match"-Bedingungen

### 2.5.1. ANY und NOTANY

Diese Prozeduren haben die Gestalt:

```
m-procedure ANY(A);
string (A);
m-procedure NOTANY(A);
string (A);
```

Die Prozedur ANY "matcht" einen String aus einem Zeichen, das mit einem Zeichen des String A übereinstimmt.

NOTANY dagegen "matcht" einen String aus einem Zeichen, das mit keinem Zeichen des String A übereinstimmt.

```
BEISPIEL:  array A, B [1:10];
           ASS (B, '('AEIOU')');
           MAT(A,ANY(B),NOTANY(B), ANY(B));
```

A wird "gematcht", wenn es in A eine Folge Vokal, Konsonant, Vokal gibt.

### 2.5.2. SPAN und BREAK

Die Prozeduren haben die Form:

```
m-procedure SPAN(A);
string A;
m-procedure BREAK(A);
string A;
```

SPAN(a) "matcht" den String maximaler Länge, der nur aus Zeichen besteht, die in A vorkommen. BREAK(A) dagegen den String maximaler Länge, der kein Zeichen aus A enthält.

Der "match" ist bei beiden Prozeduren immer erfolgreich, evtl. wird nur der Nullstring "gematcht".

BEISPIEL:
```
CAS (B, '('12 ... 9')');
CAS (C, '('ABC ... Z')');
MAT (A, ANY(C), SPAN(CAT(B, C)));
```
Es wird in A der erste auftretende Identifier "gematcht".

### 2.5.3. TAB, RTAB und REM

Durch diese Prozeduren wird der Teilstring von der derzeitigen Cursor-Position bis zu einer bestimmten Stelle, von links bzw. rechts gezählt, "gematcht".

```
m-procedure TAB(N);
  integer N;
```
TAB(N) "matcht" von der derzeitigen Cursorposition bis einschließlich des N-ten Zeichens von rechts des Untersuchungsstrings.

BEISPIEL:
```
CAS(A,'('BOGOL44O')');
MAT(A, 2, TAB(5));
```
TAB(5) "matcht" 'GOL'
```
MAT(A, 2, RTAB(3));
```
RTAB(3) "matcht" ebenfalls 'GOL'

```
m-procedure REM;
```
REM besitzt keinen Parameter und entspricht RTAB(O), d. h. "matcht" gerade den Reststring bis zum Ende.

### 2.5.4. POS und RPOS

Mit Hilfe der Prozeduren POS und RPOS kann die derzeitige Cursorposition getestet werden. Sie "matchen" jeweils den Nullstring, wenn die Cursorposition deren Parameter entspricht.

```
m-procedure POS(N);
  integer N;
```
POS(N) "matcht" den Nullstring, wenn der Cursor zwischen dem N. und dem N+1.- Zeichen steht.

BEISPIEL: a) MAT(A, POS(0), SPAN'(' ')', POS(5));

MAT ist nur erfolgreich, wenn der String A mit genau 5 Blanks beginnt.

b) MAT(A, POS(0), ANY('(' ')'));

MAT ist hier erfolgreich, wenn A mit wenigstens einem Blank beginnt.

2.5.5. ARB, ARBNO

m-procedure ARB(A[, B]);
pattern A; stringvar B;

Der Aufruf ARB(B) "matcht" einen beliebigen Teilstring, dessen Ende das Pattern A "matcht".

BEISPIEL: ARB(N), N integer >0, "matcht" jeden String, dessen Länge ≥ N ist.

Sonderregel:
ARB(0) "matcht" jeden beliebigen String, auch den Leerstring.

A='ABCDEFG'
MAT(A'('A')', ARB('('E')'));

ARB('('E')') "matcht" hier den Teilstring 'BCDE'

Der 2. Parameter ist optional und weist den Teilstring, der zusätzlich vor dem die Bedingung A matchenden Teilstring eingefügt wurde, der Variablen bzw. dem Feld B zu.

BEISPIEL: A='ABCDEFG'
Nach MAT(A,'('A')',ARB('('E')',B), SV(C)) ist B='BCD' und C='BCDE'

Bezüglich SV siehe 2.3.1!

m-procedure ARBNO(A);
pattern A;

Der Aufruf ARBNO(A) "matcht" jede beliebige Anzahl von Teilstrings hintereinander, die jeweils den Parameter A 'matchen', einschließlich des Leerstrings. ARBNO(A) hat also mehrere Alternativen.

BEISPIEL: Habe A selbst die Alternativen $A_1$, $A_2$, $A_3$ so werden folgende zusammengesetzte Pattern hintereinander getestet:

Leerstring

$A_1$

$A_1\ A_1$

$A_1\ A_1\ A_1$

$A_1\ A_1\ A_2$

$A_1\ A_1\ A_3$

$A_1\ A_2$

$A_1\ A_2\ A_1$

$A_1\ A_2\ A_2$ usw. bis

$A_3\ A_3\ A_3$

Dabei sei davon ausgegangen, daß für alle Alternativen nach mehr als 3 Wiederholungen kein 'matching' möglich sei.

BEISPIEL:
```
real procedure P; P:=ALT('('1234')', '('123')', '('341')', '('412')');
real procedure AUS(X); code;
MAT ('(',123412341')',ARBNO(P), SV(B), DO(AUS(B)), RPOS(0));
```

liefert die Ausdrucke:

| | |
|---|---|
| 1234 | $(P_1)$ |
| 12341234 | $(P_1P_1)$ |
| 1234123 | $(P_1P_2)$ |
| 123 | $(P_2)$ |
| 123412 | $(P_2P_4)$ |
| 123412341 | $(P_2P_4P_3)$ |

### 2.5.6. LEN

```
m-procedure LEN(N);
integer N;
```

LEN(N) 'matcht' jeden Teilstring der Länge N. Für N > 0 kann in den Prozeduren MAT, ASS, ALT, PAT, RU, RS, STAR, ARBND, BAL, UNTIL, NOT auch jeweils nur N stehen.

### 2.5.7. BAL

```
m-procedure BAL(A[,B]);
pattern A; string B;
```

BAL matcht Teilstrings, die die Bedingung A erfüllen und zusätzlich bezüglich zweier Klammersymbolklassen ausgeglichen sind. Die Klammersymbolklassen können in B angegeben werden. B muß 2·n Zeichen enthalten. Die ersten n Zeichen davon gehören zur Klasse Klammer auf(KA), die folgenden zur Klasse Klammer zu(KZ).

Der durch A "gematchte" Teilstring muß die gleiche Zahl von Zeichen aus KA, wie aus KZ enthalten und zu keinem Punkt dürfen mehr Zeichen aus KZ als aus KA vorgekommen sein. Falls der Parameter B fehlt, wird B= '( )' angenommen.

BEISPIEL: Nach

```
MAT('('F(B+(C·D))-E')', '('F')', BAL(0), SV(B));
```

ist B='(B+(C·D))'

2.5.8. UNTIL

m-procedure UNTIL(A);

pattern A;

Der Aufruf UNTIL(A) 'matcht' den Teilstring bis zum ersten Auftreten von A. Der Teilstring der A 'matcht' ist dabei jedoch nicht eingeschlossen. Falls A mehrere Alternativen hat, hat UNTIL jedoch nicht mehrere Alternativen, da der erste Match mit dem kürzestens Vorstring die Bedingung erfüllt.

BEISPIEL:
```
MAT('('123456')', UNTIL(ALT('('4')', '('5')'), SV(B),
   DO(AUS(B)), FAILURE)
```
liefert die Ausdrucke:

123
23
3

```
MAT('('123456')', ALT(UNTIL('('4')'), UNTIL('('5')'), SV(B),
   DO (AUS(B)), FAILURE);
```
liefert jedoch:

1234
12345
234
2345
34
345
4
45
5

2.5.9. FAILURE, SUCCEED

m-procedure FAILURE;

FAILURE ist ein pattern, das nie erfüllt ist. Es wird benutzt um den 'Scanner' nach weiteren Alternativen zu suchen lassen.

BEISPIEL: Mode unanchored
```
MAT('('MISSISSIPI')', ALT('('IS')', '('SI')', '('IP'), '('PI')'),
   SV(B), DO(AUS(B)), FAILURE);
```
liefert die Ausdrucke:

IS
SI
IS
SI
IP
PI

Ohne FAILURE wäre die Prozedur MAT nach dem ersten gefundenen Teilstring 'IS' beendet.

m-procedure SUCCEED;

SUCCEED matcht den Nullstring, jedoch im Gegensatz zu LEN(0) hat SUCCEED beliebig viele Alternativen. Bei dem Weg zurück nach links gelangt der 'Scanner' so nie über ein SUCCEED hinaus.

SUCCEED erzeugt zusammen mit FAILURE oder bei sonst nicht erfüllbaren Bedingungen endlose Schleifen.

BEISPIEL: MAT ('('ABC')', SUCCEED, PAT (1,0), SV(B), DO(AUS(B)), FAILURE);

liefert den unendlichen Ausdruck:

```
A
AB
ABC
A
AB
ABC
:
```

2.5.10. FENCE

m-procedure FENCE;

Wenn der Scanning-Algorithmus von links nach rechts läuft, 'matcht' FENCE den Nullstring. Gleichzeitig werden jedoch alle Alternativen links von FENCE vergessen. So kommt der Scanner beim Suchen von Alternativen auf dem Weg von rechts nach links nie über FENCE hinaus. Die Anwendungen von FENCE sind vielfach.

FENCE kann im 'unanchored'-Mode einen Test wie im Mode 'anchored' erlauben und zwar, effizienter als POS(0).

BEISPIEL: Das Pattern A soll ab dem 1. Zeichen von S erfüllt sein.

```
MAT(S,FENCE,A);
```

Ferner kann getestet werden, ob das erste Auftreten von Pattern A von Pattern B gefolgt wird.

BEISPIEL: Es soll getestet werden, ob das erste Auftreten von 'A' von 'B' gefolgt wird

```
ANCHOR(0);
MAT(S, '('A')', FENCE, '('B')');
```

FENCE kann auch letzter Parameter eines ALT (2.4.1) Aufrufes sein. So können bisweilen unnötige Rückläufe des Scannars gespart werden.

2.5.11. NOT

m-procedure NOT(A);

pattern A;

NOT 'matcht' den Nullstring und zwar genau dann, wenn an dieser Cursor-Position der Pattern A nicht gematcht wird. Dies erlaubt es einfach, gewisse Bedingungen auszuschließen.

BEISPIEL: Es soll ein beliebiger String der Länge 3 'gematcht' werden, der nicht mit 'A' beginnt. Dies wird erreicht durch folgendes Pattern:

```
real procedure P; P: = PAT(NOT('('A')),3);
```

2.5.12. BOOL, DO

m-procedure BOOL (X);

boolean X;

BOOL 'matcht' genau dann erfolgreich den Nullstring, wenn die Bedingung X wahr ist.

BEISPIEL: Mit POS kann nur eine feste Position getestet werden. Soll jedoch etwa das erste von Blank verschiedene Zeichen zwischen der 7. und 10. Stelle erscheinen, so kann dies durch:

```
ANCHOR(1);
MAT(S,SPAN('('  ')'),CURS(N),BOOL(N ≥ 7∧N ≤ 10))
```

getestet werden.

m-procedure DO(X);

any X;

DO matcht immer den Nullstring. Zusätzlich wird jedoch der Parameter X aktiviert, so daß z. B. ein dort stehender Prozeduraufruf ausgeführt. Der eventuell entstehende Funktionswert wird nicht beachtet. Es muß aus syntaktischen Gründen der Aufruf einer Funktionsprozedur sein. Jedoch kann jede getrennt übersetzte Prozedur, eventuell lokal, als Funktionsprozedur deklariert werden.

BEISPIEL: Ein 'gematchter' Teilstring soll ausgegeben werden.

```
CAS(A,'('ABCDEFG')');
begin  real procedure AUS(X); code;
MAT(A, '('B')', 3, SV(B), DO(AUS(B)));
end;
```

liefert den Ausdruck: CDE

2.5.13. IP, OP

m-procedure IP (S)

string S;

IP (S) ist erfolgreich, falls an dieser Stelle der Teilstring S folgt. Der Aufruf von IP kann jedoch in allen 'Matching'-Prozeduren durch Angabe von S ersetzt werden. Intern wird dann IP (S) aufgerufen.

m-procedure OP (S)

string S;

Beim Lauf des Matching-Algorithmus kann zusätzlich auch ein anderer String aufgebaut werden. Eintragungen in diesem String erfolgen durch OP und SVOP (2.3.2). Bei OP (S) wird der Nullstring gematcht und S in den zweiten (Ausgabe-)String eingetragen. Falls der Scanner auf Grund einer später nicht erfüllten Bedingung über diese Stelle zurückläuft, wird der entsprechende Ausgabeteil wieder gelöscht. Dadurch wird eine syntaktische Analyse und Testerzeugung mit Backtrack erlaubt. Man kann so in ALGOL60 eine Art syntaxgesteuerter Makroprozessor konstruieren.

BEISPIEL: Ein gegebener String S soll umgedreht ausgedruckt werden. Das Erzeugen des umgedrehten Strings geschieht durch folgendes Pattern:

```
real procedure A;
begin array B[0:1] ;
A: = ALT(PAT(1,SV(B),A,OP(B)),RPOS(0));
end;
```

Sei S = 'ABCDE' so wird nach MAT(S,A); AUS(OBUF(0));
EDCBA

ausgedruckt. Bei Pattern A ist, solange noch ein Zeichen gelesen werden kann, die 1. Alternative richtig, die jeweils ein Zeichen liest; am Ende jedoch die 2. Alternative RPOS(0).

Dies entspricht der folgenden syntaktischen Defintion:

A :: = 1 A / leer

Da der erste Aufruf von A zuletzt beendet wird, wird auch das zuerst gelesene Zeichen zuletzt gedruckt.

## 2.6. Syntaxgesteuertes Pattern-Matching

Die Prozeduren RU, RS, STAR erlauben einen syntaxgesteuerten Programmablauf ähnlich wie in CDL[8]. Es kann der Text entsprechend einer kontext-freien Grammatik nach top-down-Verfahren bearbeitet werden.

### 2.6.1. Regelakternativen RU

m-procedure RU($B_1$, $B_2$,.., $B_n$)

pattern $B_1$,...,$B_n$;

RU liefert Erfolg, wenn eine der Bedingungen $B_1$,..., $B_n$ gematcht werden kann. Im Gegensatz zu ALT werden jedoch die restlichen Alternativen später nicht nochmal abgeprüft.

### 2.6.2. Sequentielle Terme RS

m-procedure RS ($B_1$,..., $B_n$);

pattern $B_1$,...,$B_n$;

RS liefert Erfolg, wenn die Bedingungen $B_1$ bis $B_n$ von der aktuellen Cursor-Position an fortlaufend gematcht werden. Im Gegensatz zu MAT oder PAT werden jedoch, falls eine Bedingung nicht erfüllt wird, die vorigen nicht auf Alternativen getestet.

### 2.6.3. Iterationen STAR

m-procedure STAR (B);

pattern B;

Von der aktuellen Cursor-Position an, wird so oft möglich versucht B zu matchen. STAR ist immer erfolgreich, falls B nicht erfüllt wird matcht STAR den Nullstring.

### 2.6.4. Sprung GO

m-procedure GO (I);

interger I;

GO kann aufgerufen werden in den Prozeduren PAT und RS. Es dient dazu Teile einer sequentiellen Folge von Pattern zu überspringen bzw. Schleifen aufzubauen. Sei N die Anzahl der Parameter im PAT oder RU Aufruf, dann wird mit GO (I) bewirkt:

| | |
|---|---|
| $1 \leq I \leq N$ | Sprung zu dem Parameter Nr. I |
| 0 | Keine Wirkung. Erfolgreicher Match des Nullstrings. Test des nächsten Parameters. |
| -1 | Match nicht erfolgreich. Rücklauf des Scanners (PAT) bzw. Verlassen über Ausgang FAIL bei RS. |
| -2 | Verlassen über SUCC ohne Test weiterer Parameter |
| -3 | Verlassen über FAIL ohne Suchen von Alternativen, auch bei PAT. |

RU, RS und STAR erlauben eine Syntax-Darstellung ähnlich CDL oder der Backus-Nauer-Form mit zusätzlichem Kleene-Star (Iteration).

BEISPIEL: ⟨identifier⟩:: = ⟨BU⟩ $\left\{ \langle BU \rangle \mid \langle ZI \rangle \right\}^{*}$

```
real procedure IDENTIFIER;
IDENTIFIER : = RS(ANY(BU), STAR(RU(ANY(BU), ANY(ZI))));
```

BU und ZI seien 2 Strings, die alle Buchstaben bzw. Ziffern enthalten. In diesem speziellen Fall wäre dies unter Ausnutzung der SNOBOL-Möglichkeiten auch durch:

```
IDENTIFIER : = PAT(ANY(BU), SPAN (CAT(BU, ZI)));
```

darstellbar.

Da die Elemente $B_i$ sowohl Eingabe lesen können als auch Ausgabe erzeugen können (SVOP, OP) erlaubt dies die Konstruktion von Syntax-gesteuerten Makroprozessoren.

## 2.7. Neupositionieren des Cursors

### 2.7.1. Cursor zurücksetzen BACK

m-procedure BACK;

BACK setzt den Cursor wieder zurück auf die Position, die er hatte bevor der letzte Parameter vor BACK 'gematcht' wurde. So kann z. B. ein String auf mehrere Bedingungen geprüft werden.

BEISPIEL: Es soll getestet werden, ob der String S mit einem Zeichen aus dem Durchschnitt der Zeichen in den Strings A und B beginnt: ANCHOR(1);

```
MAT(S, ANY(A), BACK, ANY(B));
```

Möchte man dagegen testen, ob das erste Zeichen aus A, das im String S vorkommt auch in B enthalten ist: ANCHOR(0);

```
MAT(S, ANY(A), BACK, FENCE, ANY(B));
```

Falls einmal ein Zeichen aus B gefunden wurde, kann wegen FENCE (2.5.10.) für den Beginn des Tests keine andere Alternative gesucht werden. Will man dagegen testen, ob im String S überhaupt ein Zeichen ausA und B enthalten ist:

```
ANCHOR (0);
MAT(S, ANY(A), BACK, ANY(B));
```

### 2.7.2. Verlängern des zu 'matchenden' Strings INPU

m-procedure INPU (A);

real oder string A;

Es wird entweder der auf A stehende bzw., falls A eine Zahl ist, entsprechend A eingelesene String an den bisherigen Teststring angefügt.

Eingelesen wird:

A = 0 Konsole (Gespräch) bzw. Fremdstring (Abschnitt), jeweils ein Satz.

0 < A < 100 Ein Satz von der Datei mit der Ger. Nr. A

A > 100 Sei A = 100 · S + G, so wird der Satz Nr. S von der Datei mit der Ger. Nr. G gelesen.

Zusammen mit IPUSH (2.7.3.) kann so ein beliebig langer Text den Test durchlaufen. Es muß nur dafür gesorgt werden, daß die Länge des Teststrings zu keinem Zeitpunkt > 594 Zeichen bzw. > 1194 Zeichen, falls kein Ausgabestring benutzt wird, ist. Der Match von INPU ist der Leerstring und erfolgreich, falls die Eingabe ohne Fehler ablief.

2.7.3. Linkshift des zu 'matchenden' Strings IPUSH

```
m-procedure IPUSH(N);
integer N;
```

Mit IPUSH(N) wird das N. + 1. Zeichen des Teststrings das erste. Der Cursor wird ebenfalls entsprechend zurückgesetzt. Falls N = 0, wird der gesamte Teststring der Leerstring und der Cursor auf 0 gesetzt. Mit INPU (2.7.2.) kann der Teststring dann wieder verlängert werden. Der Match von IPUSH ist nicht erfolgreich, falls N größer als die aktuelle Cursorposition ist.

## 2.8. Konstruktion von Pattern durch den Benutzer

Sollen bestimmte Aufrufe von m-procedure öfters vorkommen, so können sie in eine Prozedur eingekleidet und unter dem Prozedurnamen aufgerufen werden. Dies kann auch noch über Parameter modifiziert werden. Die Prozedur wird als real procedure definiert.

BEISPIEL 1: Es soll ein Pattern definiert werden, mit dem alle Namen, beginnend mit einem Buchstaben, gefolgt von Buchstaben oder Ziffern erkannt werden.

```
real procedure IDENTIFIER;
IDENTIFIER := PAT(ANY(BU), SPAN(CAT(BU,ZI)));
```

BU und ZI seien Strings, die alle Buchstaben bzw. Ziffern enthalten.

BEISPIEL 2: Soll ein Term aus Folge: '(' ⟨identifier⟩ ⟨operator⟩ ⟨identifier⟩ ')' bestehen und der Operator als Parameter angegeben werden:

```
real procedure TERM(O);
TERM := PAT('('(')', IDENTIFIER, O, IDENTIFIER, '(')');
```

BEISPIEL 3: Diese Pattern-Definitionen können auch rekursiv sein.
Beispiel: Es soll die Zeichenfolge: $A^n B$ für $n \geq 0$ erkannt werden;
A und B sollen vorgebbar sein.

```
real procedure AAB(A,B);
AAB: = ALT(B, PAT(A,AAB));
```

dies entspricht der context-freien Definition:

⟨aab⟩ ::= ⟨b⟩ | ⟨a⟩ ⟨aab⟩

Falls man den Backtrack nicht benötigt, könnte dies auch durch RU und RS definiert sein.

```
real procedure AAB(A,B);
AAB: = RU(B, RS(A, AAB));
```

In diesem Falle könnte dies natürlich auch einfacher durch AAB: = RS(STAR(A), B) definiert werden.

2.8.1. "Unevaluated Expression" UE

Hat man statt $A^n B$ hingegen $BA^n$ zu erkennen, so wäre die syntaktische Definition:

⟨baa⟩ :: = ⟨b⟩ | ⟨baa⟩ ⟨a⟩

Dies ist eine Linksrekursion. Würde man den Pattern analog wie bei AAB definieren, so würde man, falls der String diesem Pattern nicht genügt, in eine unendliche Schleife gelangen. Um dies zu verhindern, dient die Prozedur UE.

```
m-procedure UE(A);
pattern A;
```

UE 'matcht' den gleichen Teilstring wie A. Jedoch ist der Rest auf jeden Fall falsch, falls die Anzahl der noch folgenden Zeichen kleiner als die Verschachtelung der UE-Aufrufe ist.

BEISPIEL 4:

```
real procedure BAA(A,B);
    BAA: = ALT(B, UE(PAT(BAA,A)));
```

Hat man den String S = 'BC' und den Aufruf MAT(S, BAA('('A')', '('B')'), RPOS(0)); so werden nacheinander folgende Pattern aufgebaut:

```
'('B')', RPOS(0)
'('B')', '('A')', '('A')'
```

Weitere Verschachtelungen von UE sind nicht möglich, so daß jede Alternative von BAA auf Fehler läuft.

Weitere Beispiele für Pattern-Definitionen:

BEISPIEL 5:

Im folgenden Beispiel erkennt EXP eine einfache Klasse von arithmetischen Ausdrücken:

```
real procedure VAR; VAR:=ANY('('XYZ')');
real procedure ADDOP; ADDOP:=ANY('('+-')');
real procedure MULOP; MULOP:=ANY(('('* /')');
real procedure FACTOR;
```

```
FACTOR:=ALT(VAR, PAT('('(')', EXP,'(')')'));
real procedure TERM;
TERM:=ALT(FACTOR,UE(PAT(TERM,MULOP,FACTOR)));
real procedure EXP; EXP:=ALT(PAT(ADDOP, TERM),
                TERM, UE(PAT(EXP, ADDOP, TERM)));
```

BEISPIEL 6: PAIR sei als ein Pattern definiert, das alle Folgen von 2 identischen Zeichen 'matcht'.

```
real procedure PAIR;
begin array X[0:1];
PAIR: = PAT (1,SV(X),X);
end;
```

BEISPIEL 7: In BIGP soll der längste Teilstring gesucht werden, der P 'matcht' und auf BIG übergeben werden.

```
real procedure  BIGP(P,BIG);
begin array TRY[0:10]; CAS(BIG,'(")');
BIGP: = PAT(PAT(P, SV(TRY), DO(SIZE(TRY) > SIZE(BIG))), SV(BIG), FAILURE);
end;
```

Das folgende Programmstück liest einen Satz und druckt das längste Wort (IDENTIFIER siehe Beispiel 1).

```
array S, BIG[0:20];
EIN(S);
MAT(S,BIGP(IDENTIFIER, BIG));
AUS(BIG);
```

## 2.9. Adressierung der Ein- und Ausgabestrings

Diese Strings liegen in der COMMON-Zone MAT. Um sie vom Benutzer her ansprechen zu können, dienen IBUF und OBUF.

### 2.9.1. IBUF

```
string procedure IBUF(O);
```

Der Aufruf IBUF wirkt wie der Name des Feldes auf dem der Eingabestring abgelegt wird, kann also sowohl zur Ein- als auch Ausgabe benutzt werden.

### 2.9.2. OBUF

```
string procedure OBUF(O);
```

Wirkt analog auf den Ausgabestring.

IBUF und OBUF müssen mit Parameter deklariert und aufgerufen werden. Der Parameter hat jedoch keine Bedeutung.

### 2.10. Änderung des 'matching'-Modus ANCHOR

Es gibt zwei 'matching'-Mode: 'anchored' und 'unanchored'.

```
procedure ANCHOR(N);
integer N;
```

Durch-ANCHOR(0) wird der Modus 'unanchored' durch ANCHOR(1) der Modus 'anchored' eingestellt. Zu Programmanfang ist 'unanchored' eingestellt.

## 3. Stringfunktionen

Im Rahmen der gewöhnlichen Algol Type-procedures können Funktionen nur einfache Zahlen- und logische Werte als Ergebnis liefern. Im Rahmen des STRING-Systems sollen jedoch auch Strings als Funktionsergebnis geliefert werden können. Intern wird dies so gehandhabt, daß der Ergebnisstring im Freispeiche BO&FSP[11] abgelegt wird.

### 3.1. Concatenation CAT

```
string procedure CAT (A1, A2, ..., An);
pattern A1, A2, ..., An;
```

Die einzelnen Zeichenketten $A_1$ bis $A_n$ werden verkettet. CAT darf nicht rekursiv aufgerufen werden.

BEISPIEL:
```
CAS(A, CAT(A,'('END')'));
```
An den String A wird der String 'END' angehängt.

### 3.2. Stringzuweisung CAS

```
procedure CAS (A, B1, ..., Bn);
string B1, ..., Bn;   (n ≥ 1).
string var A;
```

Auf A wird die Verkettung der Strings $B_1$ ... $B_n$ abgelegt. $B_1$ bis $B_1$ darf jedoch nicht A sein. Dann muß ein CAT zwischengeschaltet werden, z. B.:

```
CAS(A, CAT(B, A));
```

vor den String A wird der String B vorgeschaltet.

### 3.3. Stringanhängen KET

```
procedure KET(A, B1, ... Bn)
stringvar A; string B1 ..., Bn;
```

Die Verkettung der Strings $B_1$ ... $B_n$ wird an A angehängt. Dies entspricht dem Aufruf:

```
CAS(A, CAT(A, B1, ..., Bn));
```

### 3.4. Indirekte Referenz IND

Bisweilen möchte man einen String ansprechen, dessen Namen wiederum als String und nicht als Algol-Variable angegeben ist. Dazu dient:

```
string procedure IND(A);
string A;
```

Das Ergebnis der Prozedur ist der unter dem pattern A abgelegte String.

```
CAS(A, '('ASTRING')');
CAS(IND(A), '('ASTRING STRING')');
```

CAS(B,A) 'ASTRING' wird auf B abgelegt.
CAS(B, IND(A)); 'ASTRING STRING' wird auf B abgelegt.

Die indirekte Referenz kann auch mehrfach benutzt werden. Falls in der Prozedur CAS der 1. Parameter ein String bzw. das Resultat einer Stringfunktion ist, wird dieser String als Name des zu untersuchenden bzw. zu ändernden Strings genommen.

BEISPIEL:
```
CAS(IND(8(8A')'), '('ASTRING')';
CAS(IND('('ATRING')'), 'ASTRING STRING'
IND('('A')') liefert 'ASTRING'
IND(IND('('A')')) liefert 'ASTRINGSTRING'
```

### 3.5. Stringarrays ARR

Etwa zur Ablage von Tabellen sind ein- oder mehrdimensionale Stringarrays praktisch. Um solche zu erstellen und darauf zuzugreifen, dient die Prozedur ARR.

Falls in Algol ein array A[ 1:N, 1:M ] definiert wird, so wird folgendermaßen abgelegt:

A(1,1) A(2,1) A(3,1) ... A(N,1) A(1,2) ... A(N,2) ... A(N,M)

Die Ablage erfolgt also spaltenweise. Für höhere Dimensionen gilt ebenso, daß bei der Ablage die vorderen Indizes schneller laufen.

Möchte man also einen Stringarray A mit den Grenzen:

$$[\, a_1{:}b_1 \;,\; a_2{:}b_2 \;,\; \ldots\ldots\ldots \;\; a_n{:}b_n \,]$$

deklarieren, so muß A wie folgt deklariert werden:

$$\underline{\text{array}}\ A\ [\, a_o{:}b_o \;,\; a_1{:}b_1 \;,\; \ldots\; a_n{:}b_n \,]$$

Die Differenz $b_o - a_o$ bestimmt die mögliche Länge der Stringelemente in A. Mögliche Anzahl der Zeichen = $(b_o - a_o - 1) \cdot 6$.

```
procedure ARR (A, I1, ..., In);
value I1, ..., In; integer I1, ..., In; array A;
```

ARR ($I_1$, ..., $I_n$) liefert dann die Adresse des Elements A[ $a_o$, $I_1$, ..., $I_n$] mit TK2 als Stringadresse.

3.6. DUPL

```
string procedure DUPL (A, N);
string A; integer N;
```

Das Ergebnis ist eine N-fache Wiederholung des Strings A;

BEISPIEL:
```
ASS(A, '(' ')');
CAS(B, DUPL(A, 50));
```
erzeugt auf B einen String aus 50 Blanks.

3.7. REPLACE

```
string procedure REPLACE (X, Y, Z);
string X, Y, Z;
```

Das Ergebnis erhält man aus dem String X wobei jedes Zeichen, das in Y vorkommt, durch das an gleicher Stelle vorkommende Zeichen in Z ersetzt. Falls die Länge von Y und Z unterschiedlich oder 0 sind, wird keine Änderung vorgenommen, dafür liefert die Funktion FAIL nachher true.

BEISPIEL:
```
ASS(A, '('01101')');
CAS(A, REPLACE (A, '('01')', '('10')'));
```
liefert das 1'er Komplement.

3.8. BLANK

```
string procedure BLANK(N); integer N;
```

Der Funktionswert ist ein String aus N Blanks.

3.9. KOMP

```
string procedure KOMP(S); string S;
```

Funktionswert ist der String, der aus S entsteht, wenn man alle Blanks entfernt.

3.10. TRENNE

```
string procedure TRENNE(S, T); string S, T;
```

Der String T muß aus einem Zeichen bestehen, dem Trenner.
Funktionswert ist der Anfang von S bis zum ersten Auftreten von T (ausschl.). Der String S wird anschließend um diesen String und T verkürzt. Falls der Trenner nicht vorkommt wird der gesamte String übergeben.

3.11. TSTR

```
string procedure TSTR(S, N ,L );
string S; integer N, L;
```

Funktionswert ist der Teilstring von S, der mit dem N.-ten ($N \geq 1$) Zeichen beginnt und L Zeichen lang ist. Falls L fehlt der gesamte Reststring ab dem N-ten Zeichen.

### 3.12. Benutzerdefinierte Stringfunktionen

Es können auch Stringfunktionen definiert werden. Dabei muß nur die Konvention beachtet werden, daß Adresse und Wert des Kopfwortes übergeben wird. Für die Operationen wie Typenkennungsetzen, Adressrechnung, Zeichenverarbeitung, siehe BOGOL-TAS [5]. Da die lokalen Variablen einer Prozedur beim Verlassen der Prozedur ihre Gültigkeit verlieren, muß das Feld, auf dem der String abgelegt wird, own, d.h. im TR440-Algol, auf einem zur Prozedur gehörenden common-Feld abgelegt werden oder in einem seperaten Freispeicher BO&FSP abgelegt werden.

Als Beispiel die Prozedur CAT (nur für Strings):

```
common CAT
array CATSTR [0:100];
real procedure CAT;
begin
PARV(1,A,AB);
PARV(2,B,BB);

AZR := REF(CATSTR [0]);
AS(AZR, AD(A,B));
TOK(AB, F6, AZR, F6, SB(A, FK(1)));
RESULT(VAL(AZR)); RETURN;
end;
```

Bezüglich der Prozeduren PARV, REF, TOK, AD, SB, FK, RESULT, RETURN, siehe BOGOL-TAS [5].

## 4. Ein- Ausgabe von Strings

Als spezielle String-EA dienen die Prozeduren EIN und AUS.

### 4.1. EIN

```
procedure EIN(A);
array oder var A;
```

Von der Normaleingabe (Fremdstring bei Abschnitt, Konsoleingabe bei Gespräch) wird eine Zeile gelesen und nach A gespeichert.

```
procedure EIN(A, G);
array oder var A;
integer G;
```

Wie oben, jedoch Gerätenummer G.

```
procedure EIN(A, G, S);
array oder var A;
integer G, S;
```

Von Gerätenummer G wird Satznummer S eingelesen. Die zugehörende Datei muß (natürlich) eine Random (RAN oder RAM)-Datei sein.

4.2. AUS

Die analogen Aufrufe zur Ausgabe sind:

```
procedure AUS(A);
string A;
```

Der mit A bezeichnete String wird auf den Normalausgabemedien (Druckerprotokoll im Abschnitt, Konsolprotokoll im Gespräch) ausgegeben.

Mit Geräte- und Satznummer sind die entsprechenden Aufrufe:

AUS(A, G) und

AUS(A, G, S).

Vor der Ausgabe wird jeweils eine neue Zeile begonnen, so daß jeder Aufruf von AUS eine Zeile ergibt. Bezüglich weiterer Prozeduren zur Ein - Ausgabe von Strings siehe das Programmpaket BODAT [10].

5. Umformung Zahlen :=: Strings

5.1. NUMV, NUMVGANZ;

Um Strings in Zahlen zu wandeln, dient:

```
real procedure NUMV (A[,L])
string A; label L;
real procedure NUMVGANZ(A[,L]); string A; label L;
```

Das Ergebnis ist der Zahlenwert des Strings A. NUMVGANZ ist dabei nur auf ganze Zahlen anwendbar. L ist ein Fehlerlabel.

5.2. STRV

Die Umkehrung liefert:

```
string procedure STRV(A)
real A;
```

Das Ergebnis ist der String, der die Zahl A darstellt.

Die Zahlen werden dabei in der Algol-Syntax eingegeben. Das String-Resultat liefert die signifikanten Stellen der Zahl. Für die Festkommadarstellung zu große oder zu kleine Zahlen werden in Gleitkommadarstellung gebracht.

```
string procedure STRVADR(X); any X;
```

Der Funktionswert ist die Adresse von X: Tetradendarstellung als String mit 6 Zeichen.

string procedure STRVHEX(W); type W;

Der Funktionswert ist die Tetradendarstellung des Wortes W, in der Form:

TK␣␣ 6 Zeichen ␣ 6 Zeichen

string procedure STRVEXAKT(N[, ANZ [, FEHL ]] );
integer N, ANZ; label FEHL;

Das Ergebnis ist der String der natürlichen Zahl N auf ANZ Ziffern mit Nullen aufgefüllt. Falls ANZ zu klein ist, wird nach FEHL gesprungen. Falls ANZ nicht vorhanden wird ANZ=6 angenommen. Falls ein Fehler auftritt und FEHL fehlt wird die Zahl mit 12 Zeichen ausgedruckt, ganz rechts ein Blank und die Zahl in den übrigen 11 Zeichen rechtsbündig.

Mit STRVEAKT können insbesondere Prozeduren zur formatierten Ausgabe aufgebaut werden.

string procedure STRVZEI(N); integer N;

STRVZEI(N) liefert als Funktionswert den String aus dem Zentralcodezeichen Nr. N.

BEISPIEL: STRVZEI(192) = '('A')'

## 5.3. MLSTRV

procedure MLSTRV(A); real A;

Die Prozedur MLSTRV dient zur Formatsteuerung bei STRV. Folgende Aufrufe und Formate sind möglich:

| | | |
|---|---|---|
| MLSTRV(N) | N > 0 | Die Zahl wird mit N-Stellen in der kürzestmöglichen Form rechtsbündig dargestellt. |
| | N < 0 | Die Zahl wird, wenn möglich, innerhalb der N-Stellen ohne Exponent ausgegeben. |
| MLSTRV(n.m) | n > 0<br>1≤m≤9<br>m ≤ n | Die Zahlen werden mit genau n Zeichen ausgegeben, davon entfallen m Zeichen auf den Dezimalpunkt und die Stellen dahinter. |
| MLSTR(100+n) | n > 0 | Wirkt wie das Format n.0, d. h. ohne Dezimalpunkt und Stellen nach dem Komma. |

# 6. Zusätzliche primitive Funktionen

## 6.1. SIZE

integer procedure SIZE(S);

string S;

SIZE(S) liefert die Länge des Strings S als Funktionswert.

## 6.2. IDENT und DIFFER

IDENT und DIFFER sind logische Funktionen, die zwei Strings auf Identität vergleichen.

boolean procedure IDENT (X,Y);

string X, Y;

```
boolean procedure DIFFER (X,Y);
string X, Y;
```

IDENT liefert true, wenn X und Y gleich sind, DIFFER dann, wenn sie nicht gleich sind.

### 6.3. TRIM

Die Prozedur TRIM dient zum Entfernen von Blanks am Anfang bzw. Ende von Strings. Folgende Aufrufe sind möglich:

```
TRIM(N);
integer N;
```

Bei der Eingabe mit EIN werden bei:

N = 0 keine Blanks

N = 1 Blanks am Anfang und Ende

N = 2 Blanks am Anfang

N = 3 Blanks am Ende

abgeschnitten.

```
TRIM(N,S);
integer N; string S;
```

Die selben Operationen werden nicht bei der Eingabe, sondern am String S vorgenommen.

## 7. Programmverzweigungen: SUCC und FAIL

Ein Durchlauf des "matching"-Algorithmus wird erfolgreich genannt, wenn ein Teilstring gefunden wird, der alle Bedingungen erfüllt. Dann wird die Common-Variable SUCCV auf true gesetzt. Ist dies nicht der Fall, wird sie auf false gesetzt. Die Prozeduren SUCC und FAIL erlauben nun eine Verzweigung in Anhängigkeit des Wertes von SUCCV.

```
boolean procedure SUCC;
liefert true falls SUCCV = true;

boolean procedure FAIL;
liefert true falls SUCCV = false;
```

BEISPIEL: Die Prozedur DIFFER könnte wie folgt geschrieben werden:

```
boolean procedure DIFFER (A,B);
begin (MAT(A, POS(0), B, RPOS(0));
DIFFER := FAIL;
end;
```

Bei der Ein-Ausgabe mit BODAT[10] wird ebenfalls die Variable SUCCV entsprechend gesetzt.

## 8. Testmöglichkeiten

Zur Überwachung des Programmablaufs beim Pattern-matching gibt es eine eingebaute Tracemöglichkeit. Ebenso kann eine vom Benutzer geschriebene Traceprocedur über Systemparameter eingeschaltet werden. Desweiteren existieren Dumpmöglichkeiten.

### 8.1. TRACE

Der Systemtrace wird durch den die Prozeduraufrufe TRACE(1), TRACE(2) eingeschaltet und durch TRACE(0) ausgeschaltet.

BEISPIEL:

```
A = 'HAUS UND LAND UND WASSER'
TRACE(1) und Mode "anchored" und
Statement: MAT(A, '('HAUS')', 0,SV(B), '('LAND')');
liefert:
MAT:
HAUS UND LAND UND WASSER
1: HAUS
2:
2:  U
2:  UN
2:  UND
2:  UND
4:  LAND
```

Kann eine Bedingung nicht "gematcht" werden, so wird der "match" der Teilstrings, soweit erfolgreich, gedumpt und anschließend FAIL gedruckt.

TRACE(2) liefert nur den Ausdruck des zu "matchenden" Strings und gegebenenfalls den Ausdruck FAIL .

Erfolgt der "Match" mit Stringersetzung (ASS), so wird ausgedruckt (Mode "unanchored").

```
CAS(A, '('LAND UND   WASSER')');
ASS(A, '('UND')', '(' - ')');
ASS:
LAND UND WASSER
1 : LAND
2 : UND
LAND - WASSER
```

### 8.2. DUMP

Mit dieser Prozedur kann ein Dump von Stringinhalten bewirkt werden. Die verschiedenen Aufrufformen sind:

DUMP ($N_1$, $A_1$, $N_2$, $A_2$, ...);

pattern $A_1$, ....., $A_K$;

string $N_1$, ....., $N_K$;

Die Strings $A_1$, ..., $A_K$ werden ausgedruckt. Als Stringnamen werden zuvor jeweils $N_1$, ..., $N_K$ gedruckt.

DUMP (1, $N_1$, $A_1$, $N_2$, $A_2$, ..., );

Die jeweiligen Strings $A_i$ werden unter ihrem Namen $N_i$ in den Dumpvektor aufgenommen.

DUMP (0, $N_1$, $A_1$, $N_2$,....,);

Die Strings werden im Dumpvektor gelöscht.

DUMP;

Alle im Dumpvektor enthaltenen Strings werden ausgedruckt.

DUMP(0);

Alle folgenden Dumpbefehle sind unwirksam; mit Ausnahme von DUMP(1);

DUMP(1);

Alle Dumpbefehle sind wieder wirksam.

DUMP(-1);

Der Dumpvektor wird gelöscht.

Nach Normierung des Systems (siehe Prozedur SNOBOL) sind die Dumpbefehle unwirksam.

## 9. Effiziente Programmierung mit BOGOL-STRING

Mit Hilfe des Matching-Algorithmus können komplizierte Textmanipulationen einfach programmiert werden. Ein mehrfaches Hin- und Herlaufen des Scanners zum Aufsuchen von Alternativen benötigt jedoch seine Zeit. Daher sollten Pattern mit Alternativen nur dort benutzt werden, wo es notwendig ist.

BEISPIEL1: Möchte man den Teilstring zwischen den 11. und dem 20. Zeichen haben, so wäre dies durch POS(10), 0, POS(20) möglich. Dabei würden jedoch für 0, 10 verschiedene Alternativen versucht, bis die richtige gefunden wird.

Einfacher durch: POS(10), TAB(20) oder durch POS(10), 10.

BEISPIEL2: Das Pattern PAT(0, '('A')') sucht eins der folgenden 'A'. Die Alternativenzahl kann jedoch zumeist verringert werden durch

PAT(BREAK('('A')'), ARBNO('('A')', BREAK('('A')')), '('A')');

Die Anzahl der Wiederholungen ist nun nur noch gleich der Zahl der übersprungenen 'A'. Oft möchte man in Wirklichkeit nur das erste 'A' bzw. es ist nur eins vorhanden, dann genügt sogar: PAT(BREAK('('A')'), '('A')') oder für ein beliebiges Pattern A statt PAT(0, A) genügt dann PAT(UNTIL (A), A);

Falls jedoch ARB benötigt wird und UNTIL nicht reicht, ist es zweckmäßig, ARB in der erweiterten Form zu benutzen. Sei A ein Pattern, so schreibt man statt 0, A zweckmäßig ARB(A). Dann werden die möglichen Alternativen für 0 in ARB abgehandelt, was schneller geht. Für 0, SV(B), A kann man ARB(A, B) schreiben.

## 10. Analogien und Unterschiede zu SNOBOL4

Um dem Kenner von SNOBOL4 die Benutzung von BOGOL-STRING leichter zu machen, sollen die Analogien und Unterschiede aufgezeigt werden.

Die definierten Pattern von SNOBOL4 haben den gleichen Namen und laufen genauso ab. Insgesamt gibt es folgende Abweichungen:

(1) SPAN ist immer erfolgreich, notfalls wird der Leerstring 'gematcht'. Dem SNOBOL4-SPAN entspricht PAT(ANY(X), SPAN(X)).

(2) Für FAIL ist der Name FAILURE zu nehmen.

(3) Dem SNOBOL4 ARB entspricht der Aufruf ARB(0). ARB hat hier noch zusätzliche Leistungen.

(4) BAL ist nicht nur auf Strings, sondern auf beliebige Pattern anwendbar. Außerdem können die Klammersymbole angegeben werden.

(5) ABORT ist nicht implementiert.

(6) 'Unevalueted Expression' werden durch Prozeduren definiert. Lediglich bei Linksrekursionen muß UE benutzt werden.

(7) BOGOL-STRING arbeitet immer im Fullscan-Mode.

(8) Wertzuweisung während des Pattern Matching findet immer in Form des 'Immediate Value Assignment' statt. 'Conditional Value Assignment' ist nicht implementiert. Für P$V wird P,SV(B) geschrieben.

(9) EVAL kann durch Prozedurdefinition simuliert werden (Call by name).

(10) APPLY ist nicht implementiert.

(11) Die Operatoren Negation ( ¬ ) und Interrogation ( ? ) können durch BOOL simuliert werden.

(12) Statt NULL kann LEN(0) genommen werden.

(13) Tabellen können durch indirekte Referenz (IND) und Stringfelder durch mehrdimensionale Arrays und ARR definiert werden. Konstruktion weiterer Datenstrukturen, siehe BOGOL-DATA.

(14) Die Programmflußanweisung (S(LABEL) und F(LABEL) werden durch if SUCC then goto LABEL beziehungsweise if FAIL then goto LABEL simuliert. In Abhängigkeit von SUCC bzw. FAIL können jedoch auch andere Conditional Statements ausgeführt werden.

(15) LGT ist nicht implementiert.

(16) Die Variablen INPUT und OUTPUT existieren nicht mit der Wirkung wie in SNOBOL4.

Erweiterungen von SNOBOL4 sind ferner die Prozeduren: SVOP, OSV, RU, RS, STAR, BACK, INPU, IPUSH, UNTIL, NOT, DO, OP, IBUF, OBUF.

LITERATURVERZEICHNIS


1 TR 440 ALGOL 60 - Sprachbeschreibung

TR 440 Unterlagensammlung 440.D1.04

2 Backus, J.W. et.al.:

Revised Report on the Algorithmic Language Algol 60
Num. Math. 4(1963), p. 420
Comm. ACM 6(1963), pp. 1-17

3 Forber, D.J., R.E. Griswold and F.P. Polonsky:

SNOBOL, A String Manipulation Language
JACM, 11(1964), p. 21-30

4 Griswold, R.E., J.F. Poage, I.P. Polansky:

The SNOBOL 4 Programming Language
Prentice-Hall, Englewood Cliffs, New Jersey, 1970

5 Rosendahl, M.:

BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440.
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum Nr. 7206, 1972.

6 Hauenschild, M. und Rosendahl, M.:

BOGOL-DATA, Konstruktion von ALGOL68 Datenstrukturen in ALGOL 60.
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum.
Erscheint demnächst.

7 PREKOM, Prekompiler für ALGOL60 zur Einsetzung von Deklarationen. Programmbeschreibung, Rechenzentrum Ruhr-Universität Bochum. Programmbeschreibung erscheint demnächst.

8 Koster, C.H.A.:

Using the CDL Compiler-Compiler in Compiler Construction, Lecture Notes in Computer Science, 1974, S. 366 - 426

9 Koster, C.H.A.:

A compiler compiler
Mathematisch Centrum Amsterdam, MR 127, 1971

10 Buchmann, R.:

BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL60 und FORTRAN
Arbeitsberichte des Rechenzentrums der Ruhr-Universität Bochum Nr. 7405, 1974

11 BO&FSP, Dynamische Freispeicherverwaltung
Rechenzentrum Ruhr-Universität Bochum, Programmbeschreibung BO.E2.10

12 Koster, C.H.A.:

Affix grammars, Fu: Peck., J.E.L. (Ed.): ALGOL68
Implementation, Amsterdam - North-Holland 1971

BEISPIELE:

(1) In einem Algol-Programm sollen bei allen 2-dimensionalen Feldern die Zeilen und Spalten vertauscht werden.

Beispiel: A[B[3, 4], C[5]] := D[6,7]

soll umgewandelt werden in:

A[C[5], B[4, 3]] := D[7, 6]

Es muß also jede Folge: [<A>, <B>] umgewandelt werden in [<B>, <A>]. Die Textfolgen <A>und <B>müssen dabei bezüglich [ und ] ausgeglichen sein.

Die Prozedur BRE(X) wird daher erfüllt durch einen bezüglich [ und ] ausgeglichenen String der bis zum einen Zeichen aus X reicht.

Jede befundene Folge[<A>, <B>]wird zunächst ersetzt durch ?<B>, <A> &.

Am Ende wird dann ? durch [ und & durch ] ersetzt.

```
0819          LF51-1  RUSEND.AIDA  061928 FKZ=          09.12.75 MV170012

□QAUSGABE
 1 KAPITEL      = SND
 2 INFORMATION  = ST3
 3 MODUS        = KU

START DAUSGABE (14.04)

SAETZE DER DATEI SND (0001.00) IN DATENBASIS &STDB


007030   'BEGIN'
007040   'ARRAY'A,B,S[0:200];
007050   'ARRAY'C[0:200];
007060   'PROCEDURE' ASS(X);    'CODE';
007070   'PROCEDURE' EIN(X);    'CODE';
007080   'PROCEDURE' AUS(X);    'CODE';
007090    'PROCEDURE' CAS(X);    'CODE';
007100   'BOOLEAN''PROCEDURE'SUCC:'CODE';
007110   'PROCEDURE'TRACE(X);'CODE';
007120   'REAL''PROCEDURE' BREAK(X);    'CODE';
007130   'REAL''PROCEDURE'    SV(X);    'CODE';
007135   'PROCEDURE'ANCHOR(X);'CODE';
007140   'REAL''PROCEDURE'    CAT(X);    'CODE';
007150   'REAL''PROCEDURE'ANY(X);'CODE';
007160   'REAL''PROCEDURE'BAL(X);'CODE';
007170   'REAL''PROCEDURE'PAT(X);'CODE';
007180   'REAL''PROCEDURE'ARBNO(X);'CODE';
007190   'REAL''PROCEDURE'BRE(X);
007200   BRE:=BAL(PAT(BREAK(X),ARBNO(PAT(ANY(X),BREAK(X)))),'('□')');
007210   ANCHOR(1);
007220   TRACE(1);
007230   CC:EIN(S,1);
007240      'IF' 'NOT'SUCC 'THEN' 'GOTO' EE;
007250   AA:ASS(S,BRE  ('('□')'),SV(C),    '('□')',BRE  ('(',')'),SV(A),
007260         '(',')',BRE  ('('□')'),SV(B),'('□')',
007270   CAT(C,'('?')',    B,'(',')',A,'('&')'));
007280      'IF' SUCC 'THEN' 'GOTO' AA;
007290   BB:ASS(S,BREAK('('&')'),SV(C),'('&')',CAT(C,'('□')'));
007300      'IF' SUCC 'THEN' 'GOTO' BB;
007310   DD: ASS(S,BREAK('('?')'),SV(C),'('?')',CAT(C,'('□')'));
007320   'IF'SUCC'THEN''GOTO'DD;
007330      AUS(S,2);
007340      'GOTO'CC;
007350   EE:'END';

ENDE DAUSGABE (14.04)  0.24
```

```
9819          LF51-1  ROSEND.AIDA  081926 FKZ=          09.12.75 MV17001

@STARTE
 1 PROGRAMM      = STOHP
 2 LAUF          = -STD-
 4 DUMP          = A-NEST
 6 DATEI         = 1-D1'2-D2
 8 AKTIV         = ALLE

START STOHP
ASS:
A[B[3,4],C[5]]:=D[6,7];
1 :
  1 :A
  2 :
2 :A
4 :[
  1 :B[3
  2 :
    1 :,
    2 :4]
  2 :,4]
5 :B[3,4]
7 :,
  1 :C[5
  2 :
    1 :]
    2 :
  2 :]
8 :C[5]
10 :]
11 :
A?C[5],B[3,4]&:=D[6,7];
ASS:
A?C[5],B[3,4]&:=D[6,7];
1 :
  1 :A?C
  2 :
2 :A?C
4 :[
  1 :5]
  2 :
    1 :,
    2 :B[3
  2 :,B[3
    1 :,
    2 :4]&:=D[6
  2 :,B[3,4]&:=D[6
    1 :,
    2 :7];
  2 :,B[3,4]&:=D[6,7];
    1 :[
    2 :5],B
  2 :[5],B
2 :A?C[5],B
4 :[
  1 :3
  2 :
5 :B
7 :,
  1 :4
  2 :
8 :4
```

```
0819          LF51-1  RUSEND,AIDA  081928 FKZ=          09.12.75 MV170012
10 :]
11 :
A?C[5],B?4,3&&:=D[6,7];
ASS:
A?C[5],B?4,3&&:=D[6,7];
1 :
  1 :A?C
  2 :
2 :A?C
4 :[
  1 :5]
  2 :
    1 :,
    2 :B?4
  2 :,B?4
    1 :,
    2 :3&&:=D[6
  2 :,B?4,3&&:=D[6
    1 :,
    2 :7];
  2 :,B?4,3&&:=D[6,7];
    1 :[
    2 :5],B?4,3&&:=D
  2 :[5],B?4,3&&:=D
2 :A?C[5],B?4,3&&:=D
4 :[
  1 :6
  2 :
5 :6
7 :,
  1 :7
  2 :
8 :7
10 :]
11 :
A?C[5],B?4,3&&:=D?7,6&;
ASS:
A?C[5],B?4,3&&:=D?7,6&;
1 :
  1 :A?C
  2 :
2 :A?C
4 :[
  1 :5]
  2 :
    1 :,
    2 :B?4
  2 :,B?4
    1 :,
    2 :3&&:=D?7
  2 :,B?4,3&&:=D?7
    1 :,
    2 :6&;
  2 :,B?4,3&&:=D?7,6&;
    1 :[
    2 :5],B?4,3&&:=D?7,6&;
  2 :[5],B?4,3&&:=D?7,6&;
2 :A?C[5],B?4,3&&:=D?7,6&;
**FAIL**
ASS:
A?C[5],B?4,3&&:=D?7,6&;
1 :
```

```
0819          LF51-1  ROSEND.AIDA  081928 FKZ=          09.12.75 MV1700
2 :A?C[5],B?4,3
4 :&
5 :
A?C[5],B?4,3]&:=D?7,6&;
ASS:
A?C[5],B?4,3]&:=D?7,6&;
1 :
2 :A?C[5],B?4,3]
4 :&
5 :
A?C[5],B?4,3]]:=D?7,6&;
ASS:
A?C[5],B?4,3]]:=D?7,6&;
1 :
2 :A?C[5],B?4,3]]:=D?7,6
4 :&
5 :
A?C[5],B?4,3]]:=D?7,6];
ASS:
A?C[5],B?4,3]]:=D?7,6];
1 :
2 :A?C[5],B?4,3]]:=D?7,6];
**FAIL**
ASS:
A?C[5],B?4,3]]:=D?7,6];
1 :
2 :A
4 :?
5 :
A[C[5],B?4,3]]:=D?7,6];
ASS:
A[C[5],B?4,3]]:=D?7,6];
1 :
2 :A[C[5],B
4 :?
5 :
A[C[5],B[4,3]]:=D?7,6];
ASS:
A[C[5],B[4,3]]:=D?7,6];
1 :
2 :A[C[5],B[4,3]]:=D
4 :?
5 :
A[C[5],B[4,3]]:=D[7,6];
ASS:
A[C[5],B[4,3]]:=D[7,6];
1 :
2 :A[C[5],B[4,3]]:=D[7,6];
**FAIL**

ENDE STOHP  1.02
```

```
0619          LF51-1  ROSEND,AIDA  081928 FKZ=          09.12.75 MV170012

⊐KKOPIERE
 1 DATEI          = D1
 2 ZEILE          = 1-999999
   000010  A[B[3,4],C[5]]:=D[6,7];

ENDE TKOPIERE (6,17)  0.07
⊐KKOPIERE
 1 DATEI          = D2
 2 ZEILE          = 1-999999
   000010  A[C[5],B[4,3]]:=D[7,6];

ENDE TKOPIERE (6,17)  0.06
⊐PRINT
```

(2) Die wesentlichen in dem Bericht angeführten Beispiele sind mit TRACE durchgerechnet.

```
005010  'COMMON'MAT
005020    'ARRAY'ZZZ[1:11],IBUFFER,OBUFFER[0:99];
005030  'BEGIN'
005032  'PROCEDURE'AUSS;'IF'SUCC'THEN'AUS(<<SUCC>>)'ELSE'AUS(<<FAIL>>);
005040  'REAL''PROCEDURE'BAA(X);
005050  BAA:=ALT('('B')',PAT(UE(BAA(0)),'('A')'));
005060  'ARRAY'A,S,B,C[0:10];
005070  'REAL''PROCEDURE'OSV(X);'CODE';
005072  'REAL''PROCEDURE'OBUF(X);'CODE';
005074  'REAL''PROCEDURE'BACK;'CODE';
005080  'REAL''PROCEDURE'ARBNO(X);'CODE';
005090  'REAL''PROCEDURE'NOT(X);'CODE';
005100  'REAL''PROCEDURE'ANY(X);'CODE';
005110  'REAL''PROCEDURE'STAR(X);'CODE';
005120  'REAL''PROCEDURE'BREAK(X);'CODE';
005130  'PROCEDURE'CAS(X);'CODE';
005140  'REAL''PROCEDURE'RPOS(X);'CODE';
005150  'REAL''PROCEDURE'UE(X);'CODE';
005160  'PROCEDURE'ASS(X);'CODE';
005162  'REAL''PROCEDURE'SIZE(X);'CODE';
005164  'REAL''PROCEDURE'IBUF(X);'CODE';
005166  'REAL''PROCEDURE'SPAN(X);'CODE';
005170  'REAL''PROCEDURE'RS(X);'CODE';
005180  'REAL''PROCEDURE'FENCE ;'CODE';
005190  'REAL''PROCEDURE'RU(X);'CODE';
005195  'REAL''PROCEDURE'UNTIL(X);'CODE';
005200  'REAL''PROCEDURE'OP(X);'CODE';
005220  'PROCEDURE'MAT(X);'CODE';
005230  'PROCEDURE'AUS(X);'CODE';
005240  'PROCEDURE'ANCHOR(X);'CODE';
005260  'PROCEDURE'TRACE(X);'CODE';
005270  'REAL''PROCEDURE'FAILURE   ;'CODE';
005280  'REAL''PROCEDURE'ALT(X);'CODE';
005282  'REAL''PROCEDURE'BOOL(X);'CODE';
005290  'REAL''PROCEDURE'BAL(X) ;'CODE';
005295  'REAL''PROCEDURE'POS(X);'CODE';
005300  'REAL''PROCEDURE'PAT(X);'CODE';
005310  'REAL''PROCEDURE'SV(X);'CODE';
005312  'REAL''PROCEDURE'DO(X);'CODE';
005314  'BOOLEAN''PROCEDURE'SUCC;'CODE';
005320  'REAL''PROCEDURE'ARB(X) ;'CODE';
005400  TRACE(1);
005500  CAS(S,<<123>>);
005510  MAT(S,PAT(POS(0),0,SV(B),RPOS(0)),PAT(OP(<<[>>),OP(B),OP(<<]>>)),
005520  OSV(B));
005530  AUS(B);
005540  CAS(A,<<ABCDEFG>>);
005550  MAT(A,<<A>>,ARB(<<E>>),SV(B));
005560  AUS(B);
005570  MAT(A,<<A>>,ARB(<<E>>,B),SV(C));
005580  AUS(B);AUS(C);
```

```
0358          LF51=1  ROSEND,AIDA  035825 FKZ=          08.12.75 MV170011   RZ U

SAETZE DER DATEI SND (0001,00) IN DATENBASIS &STDDB

005590  'BEGIN'
005600  'REAL''PROCEDURE'AUS(X);'CODE';
005610  'REAL''PROCEDURE'P;P:=ALT(<<1234>>,<<123>>,<<341>>,<<412>>);
005620  MAT(<<123412341>>,ARBNO(P),SV(B),DO(AUS(B)),RPOS(0));
005630  MAT(<<MISSISSIPI>>,ALT(<<IS>>,<<SI>>,<<IP>>),SV(B),DO(AUS(B)),FAILURE)
005640  MAT(<<123456>>,UNTIL(ALT(<<4>>,<<5>>)),SV(B),DO(AUS(B)),FAILURE);
005650  'END';
005660  MAT(<<F(B*(C*D))=E>>,<<F>>,BAL(0),SV(B));
005670  AUS(B);
005680  ANCHOR(0);
005690  MAT(<<123>>,FENCE,<<2>>);
005700  AUSS;
005720  MAT(<<XACABD>>,<<A>>,FENCE,<<B>>);
005730  AUSS;
005740  'BEGIN'  'REAL''PROCEDURE'P;P:=PAT(NOT(<<A>>),3);
005742  MAT(<<1ABC1BCD>>,<<1>>,P,SV(B));AUS(B);
005744  'END';
005746  ANCHOR(1);
005748  'BEGIN' 'REAL''PROCEDURE'A;
005750   'BEGIN' 'ARRAY'B[0:1];
005752   A:=ALT(PAT(1,SV(B),A,OP(B)),RPOS(0));
005754   'END';
005755  TRACE(1);
005756  MAT(<<ABCDE>>,A);AUS(OBUF(0));
005758  'END';
005760  ANCHOR(0);
005762  'BEGIN'
005764  'REAL''PROCEDURE'IDENT;IDENT:=RS(ANY(BU),STAR(RU(ANY(BU),ANY(ZI))));
005766  'ARRAY'BU[0:20],ZI[0:10];CAS(BU,<<ABCDEFGHIJKLMNOPQRSTU>>);
005768  CAS(ZI,<<0123456789>>);
005770  MAT(<<12A34B+7>>,IDENT,SV(B));AUS(B);
005772  'END';
005774  MAT(<<ABC>>,ANY(<<ABC>>),BACK,ANY(<<BXY>>),SV(B));AUS(B);
005776  MAT(<<12A45>>,ANY(<<ABC>>),BACK,FENCE,ANY(<<AY>>),SV(B));AUS(B);
005778  MAT(<<12A4>>,ANY(<<14A>>),BACK,FENCE,ANY(<<A2>>));AUSS;
005780  MAT(<<12A4>>,ANY(<<14A>>),BACK,ANY(<<A2>>),SV(B));AUS(B);
005782  'BEGIN'
005784  'REAL''PROCEDURE'AAB(A,B);
005786  AAB:=ALT(B,PAT(A,AAB(A,B)));
005788  MAT(<<XYAAB>>,AAB(<<A>>,<<B>>),SV(B));AUS(B);
005790  MAT(<<XYZ>>,AAB(<<A>>,<<B>>));AUSS;
005792  'END';
005794  'BEGIN'
005796  'REAL''PROCEDURE'AAB;
005798  AAB:=RU(<<B>>,RS(<<A>>,AAB));
005800  MAT(<<XYAAB>>,AAB,SV(B));AUS(B);
005802  MAT(<<XY>>,AAB);AUSS;
005804  'END';
005806  'BEGIN'
005808  'REAL''PROCEDURE'AAB;AAB:=RS(STAR(<<A>>),<<B>>);
005810  MAT(<<XYAABZ>>,AAB,SV(B));AUS(B);
005812  MAT(<<XZ>>,AAB);AUSS;
005814  'END';
005815  AAA;
005816  'BEGIN'
005818  'REAL''PROCEDURE'VAR;VAR:=ANY(<<XYZ>>);
005820  'REAL''PROCEDURE'ADDOP;ADDOP:=ANY(<<+->>);
005822  'REAL''PROCEDURE'MULOP;MULOP:=ANY(<<*/>>);
005824  'REAL''PROCEDURE'FACTOR;FACTOR:=ALT(VAR,PAT(<<(>>,EXP,<<)>>));
005826  'REAL''PROCEDURE'TERM;TERM:=ALT(FACTOR,UE(PAT(TERM,MULOP,FACTOR)));
```

```
358          LF51-1  ROSEND,AIDA  035825 FKZ=          08.12.75 MV170011   RZ UN

SAETZE DER DATEI SND (0001,00) IN DATENBASIS &STODB

005828  'REAL''PROCEDURE'EXP;EXP:=ALT(PAT(ADDOP,TERM),TERM,UE(PAT(EXP,ADDOP,TER
        )));
005829  ANCHOR(1);TRACE(1);
005830  MAT(<<X*(Y+X)>>,EXP,RPOS(0));AUSS;
005834  MAT(<<X+(Y+X>>,EXP,RPOS(0));AUSS;
005836  'END';
005837  ANCHOR(0);
005838  'BEGIN'
005840  'REAL''PROCEDURE'BAA(A,B);
005842  BAA:=ALT(B,UE(PAT(BAA(A,B),A)));
005844  TRACE(1);
005846  MAT(<<BC>>,BAA(<<A>>,<<B>>),RPOS(0));
005850  'END';
005852  'BEGIN'
005854  'REAL''PROCEDURE'PAIR;
005856  'BEGIN' 'ARRAY'X[0:1];
005858  PAIR:=PAT(1,SV(X),X);
005860  'END';
005861  'REAL''PROCEDURE'AUS(X);'CODE';
005862  ANCHOR(0);
005863  MAT(<<XZZASSDFFG>>,PAIR,SV(B),DO(AUS(B)),FAILURE);
005864  'END';'BEGIN'
005865  'REAL''PROCEDURE'BIGP(P,BIG);
005867  'BEGIN'
005869  'ARRAY'TRY[0:10];CAS(BIG,<<>>);
005871  BIGP:=PAT(0,PAT(P,SV(TRY),BOOL(SIZE(TRY)>SIZE(BIG))),SV(BIG),FAILURE);
005873  'END';
005876  'ARRAY'BIG[0:10];
005877  ANCHOR(1);TRACE(1);
005878  MAT(<<A1B123C23D1234FG>>,BIGP(SPAN(<<1234>>),BIG));AUS(BIG);
005880  'END';
005882  TRACE(1);
005883  ANCHOR(1);
005884  MAT(<<1A23A45A>>,PAT(0,<<A>>),RPOS(0));
005886  MAT(<<1A23A45A>>,PAT(BREAK(<<A>>),ARBNO(PAT(<<A>>,BREAK(<<A>>))),<<A>>)
        RPOS(0));
005888  MAT(<<123456>>,OP(<<ABCD>>));
005890  AUS(IBUF(0));AUS(OBUF(0));
005990  'END';

ENDE DAUSGABE (14,04)  0,77
```

```
0358          LF51-1  ROSEND,AIDA  035825 FKZ=          08.12.75

¤STARTE
 1 PROGRAMM      = STDHP
 2 LAUF          = -STD-
 4 DUMP          = A-NEST
 8 AKTIV         = ALLE

START STDHP
MAT:
123
1  :
   1  :
   2  :
   2  :1
   2  :12
   2  :123
   4  :
2  :123
   1  :
   2  :
   3  :
3  :
[123]
MAT:
ABCDEFG
1  :
2  :A
3  :BCDE
BCDE
MAT:
ABCDEFG
1  :
2  :A
3  :BCDE
E
BCDE
MAT:
123412341
1  :
2  :

4  :
2  :1234
1234
4  :
2  :12341234
12341234
4  :
2  :1234123
1234123
4  :
2  :123
123
4  :
2  :123412
123412
4  :
2  :123412341
123412341
4  :
5  :
MAT:
```

Programmzeilen

5510

5550

5570

5620

5630

```
0358          LF51-1  ROSEND.AIDA  035825 FKZ=          08.12.75

MISSISSIPI
1  :
1  :M
2  :IS
IS
4  :
1  :MI
1  :MIS
2  :SI
SI
4  :
1  :MISS
2  :IS
IS
4  :
1  :MISSI
1  :MISSIS
2  :SI
SI
4  :
1  :MISSISS
2  :IP
IP
4  :
1  :MISSISSI
1  :MISSISSIP
1  :MISSISSIPI
**FAIL**                                        5640
MAT:
123456
1  :
2  :1234
1234
4  :
1  :1
2  :234
234
4  :
1  :12
2  :34
34
4  :
1  :123
2  :4
4
4  :
1  :1234
2  :5
5
4  :
1  :12345
1  :123456
**FAIL**                                        5660
MAT:
F(B+(C*D))=E
1  :
2  :F
3  :(B+(C*D))
(B+(C*D))
MAT:
123
1  :
```

```
2 :
**FAIL**
FAIL
MAT:                                    5720
XACABD
1 :
1 :X
2 :A
3 :
**FAIL**
FAIL
MAT:                                    5742
1ABC1BCD
1 :
2 :1
1 :1
1 :1A
1 :1AB
1 :1ABC
2 :1
  1 :
  2 :BCD
3 :BCD
BCD
MAT:                                    5756
ABCDE
  1 :A
    1 :B
      1 :C
        1 :D
          1 :E
          3 :
          4 :
        3 :E
        4 :
      3 :DE
      4 :
    3 :CDE
    4 :
  3 :BCDE
  4 :
1 :ABCDE
EDCBA
MAT:                                    5770
12A34B+7
1 :
1 :1
1 :12
  1 :A
  2 :A34B
2 :A34B
A34B
MAT:                                    5774
ABC
1 :
2 :A
1 :A
2 :B
4 :B
B
MAT:                                    5776
12A45
```

```
0358          LF51-1  ROSEND,AIDA  035825 FKZ=          08.12.75

1  :
1  :1
1  :12
2  :A
4  :
5  :A
A
MAT:                                    5778
12A4
1  :
2  :1
4  :
**FAIL**
FAIL
MAT:                                    5780
12A4
1  :
2  :1
1  :1
1  :12
2  :A
4  :A
A
MAT:                                    5788
XYAAB
1  :
1  :X
1  :XY
   1  :A
      1  :A
      2  :B
   2  :AB
2  :AAB
AAB
MAT:                                    5790
XYZ
1  :
1  :X
1  :XY
1  :XYZ
**FAIL**
FAIL
MAT:                                    5800
XYAAB
1  :
1  :X
1  :XY
   1  :A
      1  :A
      2  :AB
   2  :AAB
2  :AAB
AAB
MAT:                                    5802
XY
1  :
1  :X
1  :XY
**FAIL**
FAIL
MAT:                                    5810
XYAABZ
```

```
0358          LF51-1  ROSEND.AIDA  035825 FKZ=          08.12.75

1  :
   1  :
1  :X
   1  :
1  :XY
   1  :AA
   2  :AAB
2  :AAB
AAB
MAT:                                    5812
XZ
1  :
   1  :
1  :X
   1  :
1  :XZ
   1  :
**FAIL**
FAIL
MAT:                                    5830
X*(Y+X)
1  :X
   1  :X
   2  :*
      1  :(
      2  :Y
         1  :Y
            1  :Y
               1  :Y
         1  :Y
         2  :+
         3  :X
      2  :Y+X
      3  :)
   3  :(Y+X)
1  :X*(Y+X)
2  :
SUCC
MAT:                                    5834
X+(Y+X
1  :X
   1  :X
      1  :X
         1  :X
            1  :X
               1  :X
                  1  :X
   1  :X
   2  :+
      1  :(
      2  :Y
         1  :Y
            1  :Y
         1  :Y
         2  :+
         3  :X
      2  :Y+X
            1  :Y
            1  :Y
            2  :+
            3  :X
         1  :Y+X
```

```
0358          LF51-1  ROSEND.AIDA  035825 FKZ=          08.12.75

    1 :Y
      1 :Y
    1 :X
      1 :X
        1 :X
          1 :X
            1 :X
    1 :X
    2 :+
      1 :(
      2 :Y
        1 :Y
        1 :Y
        2 :+
        3 :X
      2 :Y+X
      1 :Y
      1 :X
        1 :X
          1 :X
            1 :X
      1 :X
      2 :+
        1 :(
        2 :Y
        1 :X
          1 :X
            1 :X
        1 :X
        2 :+
          1 :(
          2 :Y
          1 :X
            1 :X
          1 :X
          2 :+
            1 :(
            2 :Y
            1 :X
            1 :X
            2 :+
              1 :(
              2 :Y
**FAIL**
FAIL
MAT:                                    5846
BC
1 :
2 :B
  1 :B
    1 :B
1 :B
1 :BC
**FAIL**
MAT:                                    5863
XZZASSDFFG
1 :
  1 :X
1 :X
  1 :Z
  3 :Z
2 :ZZ
```

```
0358          LF51-1  RDSEND.AIDA  035825 FKZ=          08.12.75

ZZ
4 :
1 :XZ
  1 :Z
1 :XZZ
  1 :A
1 :XZZA
  1 :S
  3 :S
2 :SS
SS
4 :
1 :XZZAS
  1 :S
1 :XZZASS
  1 :D
1 :XZZASSD
  1 :F
  3 :F
2 :FF
FF
4 :
1 :XZZASSDF
  1 :F
1 :XZZASSDFF
  1 :G
1 :XZZASSDFFG
**FAIL**                                    5878
MAT:
A1B123C23D1234FG
  1 :
    1 :
  1 :A
    1 :1
    3 :
  2 :1
  1 :A1
    1 :
  1 :A1B
    1 :123
    3 :
  2 :123
  1 :A1B1
    1 :23
  1 :A1B12
    1 :3
  1 :A1B123
    1 :
  1 :A1B123C
    1 :23
  1 :A1B123C2
    1 :3
  1 :A1B123C23
    1 :
  1 :A1B123C23D
    1 :1234
    3 :
  2 :1234
  1 :A1B123C23D1
    1 :234
  1 :A1B123C23D12
    1 :34
```

```
  1 :A1B123C23D123
    1 :4
  1 :A1B123C23D1234
    1 :
  1 :A1B123C23D1234F
    1 :
  1 :A1B123C23D1234FG
    1 :
**FAIL**
1234
MAT:                                    5884
1A23A45A
  1 :
  1 :1
  2 :A
1 :1A
  1 :1A
  1 :1A2
  1 :1A23
  2 :A
1 :1A23A
  1 :1A23A
  1 :1A23A4
  1 :1A23A45
  2 :A
1 :1A23A45A
2 :
MAT:                                    5886
1A23A45A
  1 :1
  2 :
  3 :A
1 :1A
    1 :A
    2 :23
  2 :A23
  3 :A
1 :1A23A
    1 :A
    2 :45
  2 :A23A45
  3 :A
1 :1A23A45A
2 :
MAT:
123456
1 :
123456
ABCD

ENDE STDHP  2,93
```

Beispiel 3: Es sollen alle Teilstrings des Strings 'ELEGANTERE LEUTE', die zwischen zwei 'E' stehen, nach der Länge sortiert ausgedruckt werden.

```
0492          LF50=1  ROSEND,AIDA  049661 FKZ=          20.02.76 MV

   015010  'BEGIN'
   015020  'PROCEDURE'MAT(X);'CODE';
   015030  'REAL''PROCEDURE'SV(X);'CODE';
   015040  'PROCEDURE'ASS(X);'CODE';
   015050  'BOOLEAN''PROCEDURE'SUCC;'CODE';
   015060  'BOOLEAN''PROCEDURE'FAIL;'CODE';
   015070  'ARRAY'A[0:20];
   015080  'ARRAY'B,C,D[0:30];
   015090  'INTEGER'I,J,K,L;
   015100  'REAL'X,Y,Z;
   015110  'REAL''PROCEDURE'BREAK(X);'CODE';
   015120  'PROCEDURE'CAS(X);'CODE';
   015122  'PROCEDURE'PRINT(X);'CODE';
   015124  'PROCEDURE'TYPE(X);'CODE';
   015130  'REAL''PROCEDURE'UNT(X);
   015140  UNT:=PAT(BREAK(X),ARBNO(PAT(X,BREAK(X))));
   015150  'PROCEDURE'TRACE(X);'CODE';
   015160  'PROCEDURE'ANCHOR(X);'CODE';
   015170  'REAL''PROCEDURE'UNTIL(X);'CODE';
   015180  'REAL''PROCEDURE'PAT(X);'CODE';
   015190  'REAL''PROCEDURE'ARBNO(X);'CODE';
   015200  ANCHOR(1);
   015210  CAS(C,<<ELEGANTERE LEUTE>>);
   015220  PRINT(C);
   015230  PRINT(<<>>);
   015240  'FOR'I:=1'STEP'1'UNTIL'16'DO'
   015250  'BEGIN'
   015260  CAS(A,C);
   015270  L1: MAT(A,  UNT (<<E>>),SV(D),<<E>>,I,SV(B),<<E>>);
   015280  'IF'FAIL'THEN''GOTO'L2;
   015290  ASS(A,D,<<E>>,<<>>);
   015300  PRINT(B);
   015310  'GOTO'L1;
   015320  L2:  PRINT(<<---- >>);TYPE(I);
   015330  CAS(A,C);
   015340  'END';
   015350  'END'

ENDE KOPIERE (6,17)  0,14
```

```
0492          LF50-1  ROSEND,AIDA  049661 FKZ=          20.02.76 MV

¤STARTE
 1 PROGRAMM      = STDMP
 2 LAUF          = -STD-
 4 DUMP          = A-NEST
 8 AKTIV         = ALLE

START STDMP (1,00)

ELEGANTERE LEUTE

L
R
---- 1
 L
UT
---- 2
---- 3
GANT
RE L
---- 4
 LEUT
---- 5
LEGANT
GANTER
---- 6
RE LEUT
---- 7
LEGANTER
---- 8
GANTERE L
---- 9
---- 10
LEGANTERE L
---- 11
GANTERE LEUT
---- 12
---- 13
LEGANTERE LEUT
---- 14
---- 15
---- 16

ENDE STDMP (1,00)  1.36
¤PKOPIERE
 1 DATEI         = STELE2
 2 ZEILE         = 1-999999
```

Beispiel 4: Für die Längen I := 1 bis 4 sollen alle Teilstrings, die zwischen zwei Vokalen stehen, ausgedruckt werden.

```
0492          LF50-1  ROSEND,AIDA  049661 FKZ=          20.02.76 MV

000010  'BEGIN'
000020  'PROCEDURE'MAT(X);'CODE';
000030  'REAL''PROCEDURE'SV(X);'CODE';
000040  'PROCEDURE'ASS(X);'CODE';
000050  'BOOLEAN''PROCEDURE'SUCC;'CODE';
000060  'BOOLEAN''PROCEDURE'FAIL;'CODE';
000070  'ARRAY'A[0:20];
000080  'ARRAY'B,C,D[0:30];
000090  'INTEGER'I,J,K,L;
000100  'REAL'X,Y,Z;
000110  'ARRAY'E[0:10];
000118  'REAL''PROCEDURE'BREAK(X);'CODE';
000120  'PROCEDURE'CAS(X);'CODE';
000122  'REAL''PROCEDURE'ANY(X);'CODE';
000130  'PROCEDURE'TRACE(X);'CODE';
000132  'PROCEDURE'ANCHOR(X);'CODE';
000133  'PROCEDURE'TYPE(X);'CODE';
000134  'REAL''PROCEDURE'UNTIL(X);'CODE';
000135  'PROCEDURE'PRINT(X);'CODE';
000136  'REAL''PROCEDURE'PAT(X);'CODE';
000137  'REAL''PROCEDURE'ARBNO(X);'CODE';
000138  ANCHOR(1);
000140  CAS(C,<<ELEGANTERE LEUTE>>);
000142  CAS(E,<<AEIOU>>);
000150  PRINT(C);
000160  PRINT(<<>>);
000170  'FOR'I:=1'STEP'1'UNTIL'4'DO'
000180  'BEGIN'
000190  CAS(A,C);
000200  L1: MAT(A, 0,SV(D),ANY(E),1,SV(B),ANY(E));
000210  'IF'FAIL'THEN''GOTO'L2;
000220  ASS(A,D, 1   ,<<>>);
000230  PRINT(B);
000240  'GOTO'L1;
000250  L2:  PRINT(<<---- >>);TYPE(I);
000260  CAS(A,C);
000270  'END';
000280  'END'
```

```
0492          LF50-1  ROSEND,AIDA  049661 FKZ=          20.02.76 MV

¤STARTE
 1 PROGRAMM      = STDHP
 2 LAUF          = -STD-
 4 DUMP          = A-NEST
 8 AKTIV         = ALLE

START STDHP (1,00)

ELEGANTERE LEUTE

L
G
R
T
---- 1
NT
 L
UT
TE
---- 2
LEG
 LE
UTE
---- 3
GANT
NTER
RE L
---- 4

ENDE STDHP (1,00)  0,58
```

Bisher erschienene Arbeitsberichte des Rechenzentrums

der Ruhr-Universität Bochum

Nr. 7101: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA; eine Dialogsprache für den TR 440 (vergriffen)

Nr. 7102: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL (vergriffen)

Nr. 7103: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer (vergriffen)

Nr. 7104: 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971)

Nr. 7105: H. Wupper
WR MB02 - Ein einfaches Band-Betriebssystem für einen mittleren Rechner

Nr. 7201: H. Windauer
Existenzsätze zur (0,1,..,R-2,R) - Interpolation

Nr. 7202: W. Schelongowski
DIATRACE - Ein System zur interaktiven Assemblerprogrammierung

Nr. 7203: M. Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA

Nr. 7204: R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR 440 in der RUB (vergriffen)

Nr. 7205: 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)

Nr. 7206: M. Rosendahl
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440

Nr. 7207: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Regelmechanismus und Berechnung der Rohrströmung)

Nr. 7208: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Regelmechanismus und Berechnung der Rohrströmung)

Nr. 7209: H. Ehlich
Anregung und Kritik zum Betriebs— und Programmiersystem der TR 440

Nr. 7210: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60

Nr. 7211: H. Camici, H. Claus, H. Ehlich, D. Kipp
Arbeitsbericht über ein Programm zur Haushaltsführung

Nr. 7301: R. Mannshardt, K.-H. Mohn, H. Münch, P. Pottinger
Einführung in die Benutzung des Teilnehmer Rechensystems TR 440
2. geänderte Auflage (vergriffen)

Nr. 7302: K.-H. Mohn
Über einige Anwendungen des Computers in der Medizin

Nr. 7303: R. Buchmann
BODAT, ein schenlles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN

Nr. 7304: M. Hauenschild
Ansätze zur komplexen Kreisarithmetik

Nr. 7305: R. Buchmannn
RB&QUELLHALT, ein TR440-Datenbanksystem zur platzsparenden Quellhaltung auf Datenträgern mit direktrem Zugriff (LFD, WSP)

Nr. 7306: 6. Jahresbericht des Rechenzentrums (1.7.1972 bis 31.12.1973)

Nr. 7401: R. Buchmann
Der Systemoperator BO&BS3OP
Messungen und Steuerungen des Betriebssystems auf Operatorebene

Nr. 7402: R. Mannshardt
Herleitung und Prüfung spezieller Runge-Kutta-Verfahren mit einem impliziten Rechenschritt

Nr. 7403: R. Buchmann, H. Wupper
Unzulänglichkeiten des TR 440 Programmiersystems und ihre Umgehung

Nr. 7404: R. Green, K.-H. Mohn
Quellbezogene FORTRAN Optimierungen für den Compiler des TR 440

Nr. 7405: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN (2..ergänzte Auflage)

Nr. 7501: R. Buchmann
Zur Theorie der Montage von Programmoduln

Nr. 7502: 7. Jahresbericht des Rechenzentrums (1.1. bis 31.12.1974)

Nr. 7503: H.-D. Sander
BOTRAN, eine Fortran Sparacherweiterung durch Code-Prozeduren

Nr. 7504: W. Schelongowski
DIATRACE - Ein System zur interaktiven Assemblerprogrammierung

Nr. 7505: Camici, Prof. Dr. Ehlich, Schürmann
Über ein Programm zur Material- und Vervielfältigungs-Abrechnung

Nr. 7506: Camici, Prof. Dr. Ehlich, Herrmannies
Über ein Programm für die Telefonabrechnung einer Nebenstellenanlage

Nr. 7507: Camici, Prof. Dr. Ehlich, Kipp
Über ein Programm zur Haushaltsführung

Nr. 7508: Camici, Cipa, Prof. Dr. Ehlich
Bericht über ein Programm zu Verwaltung der Studentendaten

Nr. 7509: J. Riege
Zur mehrdimensionalen Spline-Interpolation bezüglich beliebiger linearer Funktionale

Nr. 7601: H. Ehlich, J. Riege u. K.-H. Schloßer
Ein Programmsystem zur Ausleihverbuchung und interaktiven Rechnerunterstützung in der allgemeinen Buchbestandsverwaltung
Teil 1: Offline-System

Nr. 7602: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL60
2. erweiterte und geänderte Version