Lexikalische Analyse

Aufgaben:

- Lesen des Eingabetextes und Zerlegung in die einzelnen Zeichen im Sinne der Grammatik der Programmiersprache(Morpheme oder auch Atome oder Lexeme).
- Morpheme können aufeinander folgen (a+b*77) oder durch Trennzeichen voneinander getrennt sein (call f1)
- Elliminierung von irrelevanten Textteilen (Kommentare, Leerzeichen...)
- Konvertierungen
- Erkennung von Wortsymbolen und Ersetzung durch besser handhabbare Symbolcodes
- Die Aufgabenteilung von Lexer und Parser ist verschieblich. So kann die Erkennung zusammengesetzter Symbole durch den Scanner, aber auch von der Syntaxanalyse realisiert werden. Auch der Aufbau von Symboltabellen wird mitunter schon von der lexikalischen Analyse vorbereitet.

Einordung in den Compiler

die lexikalische Analyse kann als

- selbständiger Compilerpass,
- als paralleler Prozess oder als
- Unterprogramm der Syntaxanalyse

organisiert sein. Bei Einpasscompilern, die im Rahmen der Vorlesung vorrangig diskutiert werden, soll die lexikalische Analyse ein Unterprogramm sein, das jeweils das nächste Morphem liefert.

Der Lexer selbst arbeitet in zwei Schritten.

Schritt 1:

- Aus dem Eingabestrom werden die Zeichen gelesen und in einem Puffer alle zum Morphem gehörenden Zeichen gesammelt. Hierzu wird ein Automat verwendet.
- Schritt 2:
- Nachbereitung in Abhängigkeit des letzten Zustandes des Automaten (Finalzustand).
- Schlüsselworterkennung, falls Bezeichner, das kann auch durch den Automaten geschehen, ist aber sehr aufwändig
- Konvertierung, falls Zahl
- Eintragen aller relevanten Informationen (Morphemcode, Morphemwert, Zeilen-/Spaltenposition) in eine Struktur Morphem.

Grammatik der PL/0 Token

Terminales Alphabet(T)

Ziffern (0..9)

Buchstaben (A..Z, a..z)

Sonderzeichen

```
(  + | - | * | / | , | . | ; | ( | ) | ? | ! | # | = | < | > | :  )
```

Nicht terminales Alphabet(N)

<Morphem>, <Symbol>, <Zahl>, <Zi>, <Bezeichner>, <Buzi>, <Bu>

Startsymbol(s)

<Morphem>

Regeln(R)

Regeln der Grammatik der Morpheme von PL/0 ohne Wortsymbole: (Wortsymbole werden als Bezeichner erkannt und müssen in der Nachbereitung erkannt und behandelt werden, im weiteren die favorisierte Methode.)

<Morphem> ::=<Sonderzeichen>|<Zahl>|<Bezeichner>

<Symbol> ::= + | - | * | / | ,
| . | ; | (|) | ?
| ! | # | = | < | >
| <= | >= | :=

<Zahl> ::= Zi {<Zi>}

<Zi> ::= 0 | 1 | 2 . . . 7 | 8 | 9

<Bezeichner> ::= Bu {BuZi}

<BuZi> ::= Bu | Zi

<Bu> ::= A | B | ... | Z | a | b | ... | z

Regeln der Grammatik der Morpheme von PL/0 mit Wortsymbolen:
(Wortsymbole werden durch den Automaten erkannt)

<Morphem> ::=<Sonderzeichen>|<Zahl>|<Bezeichner>

<Symbol> ::= + | - | * | / | ,
| . | ; | (|) | ?
| ! | # | = | < | <=
| > | >=| :=
| BEGIN | CALL | CONST | END | DO | IF
| PROCEDURE | THEN | VAR | WHILE

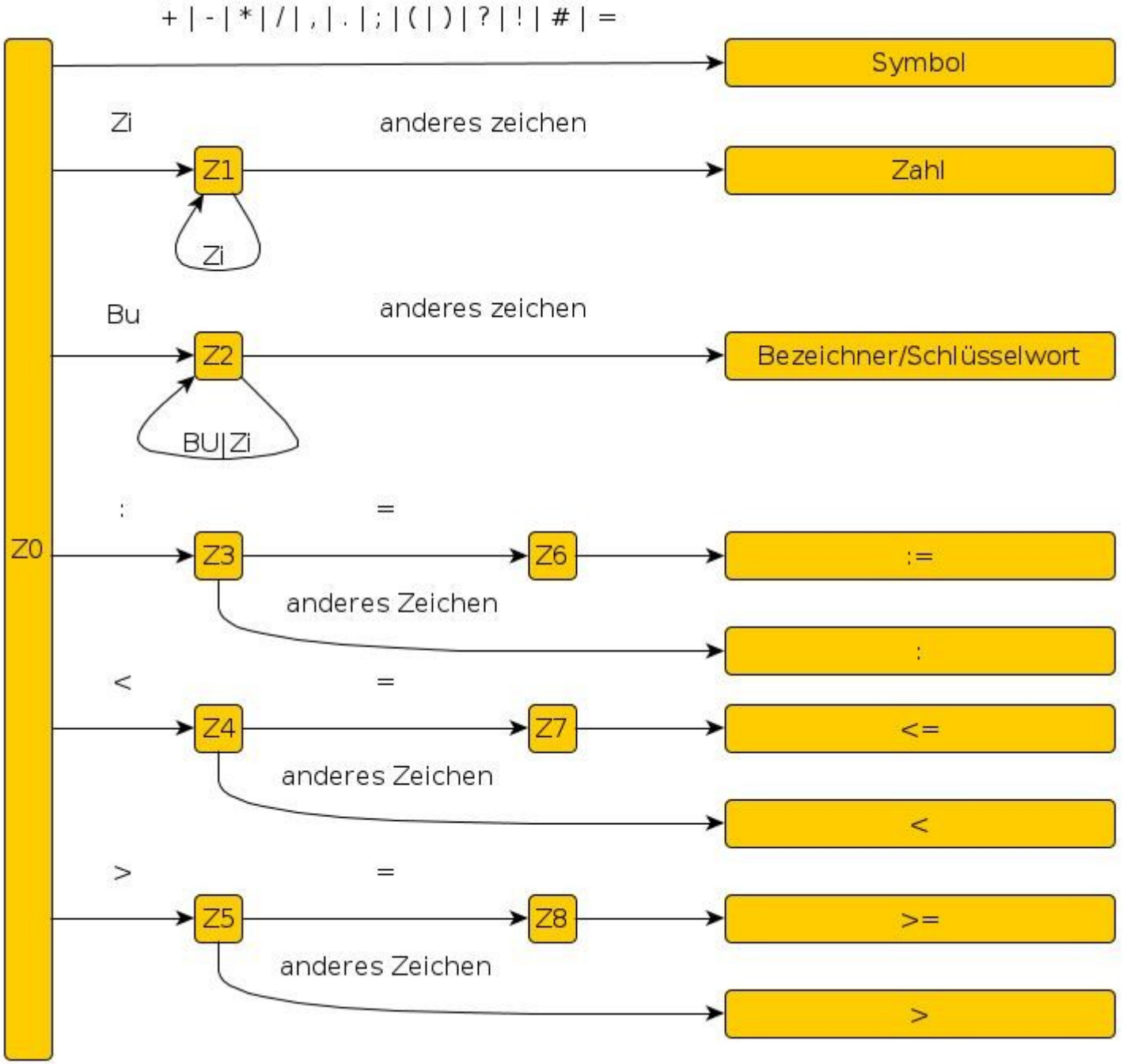
<Zahl> ::= Zi {<Zi>}

<Zi> ::= 0 | 1 | 2 . . . 7 | 8 | 9

<Bezeichner> ::= Bu {BuZi}

<BuZi> ::= Bu | Zi

<Bu> ::= A | B | ... | Z | a | b | ... | z


+ | - | * | / | , | . | ; | (|) | ? | ! | # | =
Symbol
Zi
anderes zeichen
Z1
Zahl
Zi
Bu
anderes zeichen
Z2
Bezeichner/Schlüsselwort
BU|Zi
:
=
Z0
Z3
Z6
:=
anderes Zeichen
:
<
=
Z4
Z7
<=
anderes Zeichen
<
>
=
Z5
Z8
>=
anderes Zeichen
>

Der endliche, deterministische Automat als Grundlage der lexikalischen Analyse

$A=(X,Y,Z,f,g,F,z_0)$

X: Eingabealphabet
Y: Resultatalphabet
Z: Menge der Zustände
f: Schalt-/Übergangsfunktion
g: Ausgabefunktion
F: Menge der Finalzustände
z_0: Anfangszustand

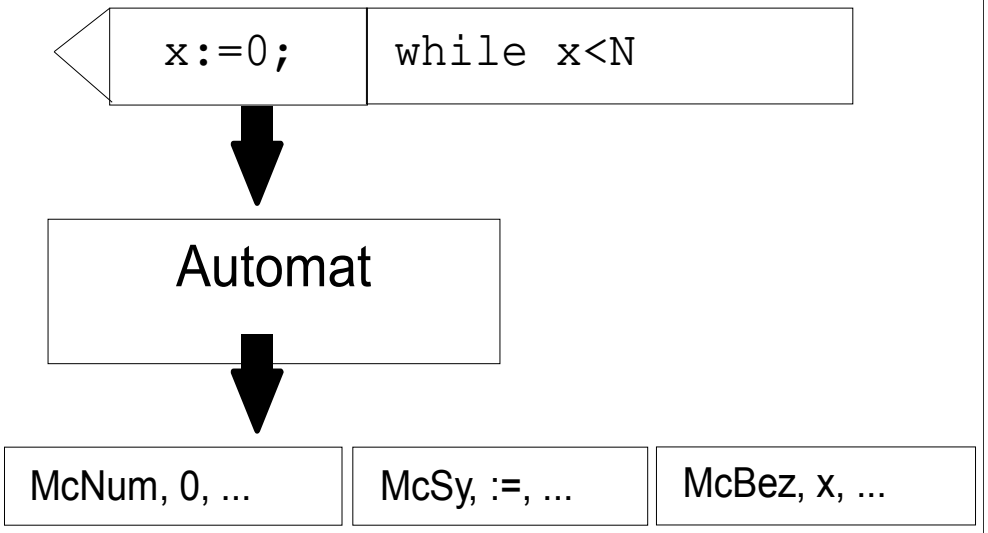


Bild 1: Automat erzeugt Morpheme

Eingabealphabet X:

Für einen PL/0 Scanner umfasst das Eingabealphabet
die Buchstaben (A-Z, a-z),
die Ziffern (0-9) sowie
eine Reihe von Sonderzeichen (+ - * / ( ) , . ; # ? ! < > : =).

Menge der Finalzustände F:

Ein Finalzustand ist dann erreicht, wenn ein Token vollständig erkannt wurde. Die zu erkennenden Token umfassen dabei sinnvollerweise:

- Sonderzeichen (Z0, Z3, Z4, Z5)
- Bezeichner / Wortsymbol (Z1)
- Zahl (Z2)
- := (Z6)
- <= (Z7)
- >= (Z8)

Resultatalphabet

- Token Sonderzeichen
- Token Bezeichner
- Token Zahl
- Token :=, :, <=, <, >=, >

Schaltfunktion

- Jeweils der Folgezustand in Abhängigkeit von Eingabezeichen und Zustand

Ausgabefunktion

- Übernahme des Eingabezeichens in den Puffer

Automatentabelle

- Tabelle bestehend aus Zeilen und Spalten, gut als zweidimensionales Array implementierbar
- Zeilen bilden die Zustände des Automaten
- Spalten werden durch die Eingabezeichen gebildet. Es ergibt sich eine sehr große Automatentabelle, auf Grund sehr vieler Spalten.
- Jede Zelle enthält den Folgezustand und die auszuführende Aktion

Zeichenklassifizierung

- Verringerung der Spalten durch Klassifizierung, Zusammenfassung gleichzubehandelnder Zeichen zu einer Zeichenklasse (zB.: Ziffern, Buchstaben...).

- Implementierungsmöglichkeiten:
  Macros aus ctype.h (isdigit, isletter ….) oder
  - Zeichenklassenvector:(Indizierung durch Ascii-Code → 128 Spalten)
  - Beschränkung auf die tatsächlich verwendeten Eingabezeichen (A..Z, a..z, 0..9, + - * / ( ) , . ; # ? ! < > : =)

Zeichenklassenvektor

- Mittels Zeichenklassenvektor kann die Zeichenklassifizierung sehr effizient vorgenommen werden.
- Der Zeichenklassenvektor hat die Länge 128, bzw. 256, wenn man den Ascii-code zugrunde legt.
- Er enthält zu jedem Ascii-code an dessen Stelle einen Wert, der die Zeichenklasse repräsentiert.

```
/* Zeichenklassenvector */
static char vZKl[128]=
/*      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F     */
/*-------------------------------------------------------*/
 /* 0*/{7, 7, 7, 7,           ...           7, 7, 7,/* 0*/
 /*10*/ 7, 7, 7, 7,           ...           7, 7, 7,/*10*/
 /*20*/ 7, 0, 0, 0,           ...           0, 0, 0,/*20*/
 /*30*/ 1, 1, 1, 1,           ...           4, 6, 0,/*30*/
 /*40*/ 0, 2, 2, 2,           ...           2, 2, 2,/*40*/
 /*50*/ 2, 2, 2, 2,           ...           0, 0, 0,/*50*/
 /*60*/ 0, 2, 2, 2,           ...           2, 2, 2,/*60*/
 /*70*/ 2, 2, 2, 2,           ...           0, 0, 0}/*70*/;
```

0: Sonderzeichen (alle oder nur erlaubte)

1 Ziffern

2 Buchstaben

3 :

4 =

5<

6>

7 Sonstige Steuerzeichen

Obige Darstellung täuscht optisch etwas, es ist ein eindimensionales Feld (Vektor). Für die bessere Lesbarkeit sind immer 16 Codes auf einer Zeile angegeben. So kann man Zeile und Spalte (hexadezimal) aus dem ASCII-code sofort ablesen. Die Kommentarspalten links und rechts erleichern die Zuordnung zusätzlich.
ASCII-code ‘A‘ =0x41, Zeichenklasse 2
ASCII-code ‘3‘ =0x33, Zeichenklasse 1

Atomatentabelle

- Die Zeilen repräsentieren die Zustände des Automaten
- Die Spalten werden durch die Eingabezeichen bzw. durch die Zeichenklassen bestimmt.
- Jede Zelle der Tabelle enthält den Folgezustand und eine Information über eine auszuführende Lese- oder Schreibfunktion.

Die Automatentabelle

Folgezustand:								
	SoZei	Ziffer	Buchstabe:	=	<	>	Sonst
Z0								
Z1								
Z2								
Z3								
Z4								
Z5								
Z6								
Z7								
Z8								

Aktion (lesen, schreiben, schreiben als Großbuchstabe, beenden)								
	SoZei	Ziffer	Buchstabe:	=	<	>	Sonst
Z0								
Z1								
Z2								
Z3								
Z4								
Z5								
Z6								
Z7								
Z8								

Die Automatentabelle

Folgezustand:								
	SoZei	Ziffer	Buchstabe	:	=	<	>	Sonst
Z0 (Symb)	e	1	2	3	e	4	5	0
Z1	e	1	e	e	e	e	e	e
Z2	e	2	2	e	e	e	e	e
Z3	e	e	e	e	6	e	e	e
Z4	e	e	e	e	7	e	e	e
Z5	e	e	e	e	8	e	e	e
Z6	e	e	e	e	e	e	e	e
Z7	e	e	e	e	e	e	e	e
Z8	e	e	e	e	e	e	e	e

Für e kann auch immer aktuelle Zustand beibehalten oder 0 eingetragen werden

Aktion (lesen, schreiben, schreiben als Großbuchstabe, beenden)								
	SoZei	Ziffer	Buchstabe	:	=	<	>	Sonst
Z0	s+l+b	s+l	sg+l	s+l	s+l+b	s+l	s+l	b
Z1	b	s+l	b	b	b	b	b	b
Z2	b	s+l	sg+l	b	b	b	b	b
Z3	b	b	b	b	s+l	b	b	b
Z4	b	b	b	b	s+l	b	b	b
Z5	b	b	b	b	s+l	b	b	b
Z6	b	b	b	b	b	b	b	b
Z7	b	b	b	b	b	b	b	b
Z8	b	b	b	b	b	b	b	b

Die Automatentabelle

- Zusammenfassung beider Tabellen
- Der Folgezustand bei Beenden ist unerheblich, da jeder neue Aufruf des Lexers in Zustand 0 beginnt
- Der Zustand, in dem der Lexer die Funktion Beenden ausführt, entspricht dem Finalzustand

Aktion (lesen, schreiben, schreiben als Großbuchstabe, beenden)								
	SoZei	Ziffer	Buchstabe	:	=	<	>	Sonst
Z0	s+l+b	1 s+l	2 sg+l	3 s+l	s+l+b	4 s+l	5 s+l	b
Z1	b	1 s+l	b	b	b	b	b	b
Z2	b	2 s+l	2 sg+l	b	b	b	b	b
Z3	b	b	b	b	6 s+l	b	b	b
Z4	b	b	b	b	7 s+l	b	b	b
Z5	b	b	b	b	8 s+l	b	b	b
Z6	b	b	b	b	b	b	b	b
Z7	b	b	b	b	b	b	b	b
Z8	b	b	b	b	b	b	b	b

Die Aktionen

Schreiben, lesen, beenden	`void fslb(void);`
Schreiben, lesen	`void fsl(void);`
Schreiben Großbuchstabe, lesen	`void fgl(void);`
Beenden	`void fb(void);`
Lesen	`void fl (void);`

Die Aktionen bestehen wiederum aus den elementaren Aktionen

- Lesen
- Schreiben
- Schreiben als Großbuchstabe
- Beenden

Jede Zelle der Automatentabelle enthält nun

den jeweiligen Folgezustand und

Einen Verweis auf eine auszuführende Aktion

Implementationsmöglichkeiten sind

Objekte mit 2 Elementen (Array oder Struktur) für

- Folgezustand
- Funktion

Der Verweis auf die Funktion kann

- durch eine Funktionsnummer oder
- als Funktionspointer

implementiert werden.

Variablen des Beispiellexers

Variable	Verwendung	Wert bei Aufruf
char x;	aktuelles Eingabezeichen	Akt. Eingabezeichen
char z;	Zustand des Automaten	0
char zx;	Folgezustand des Automaten	0
char end;	Flag über das Beenden	0
tMorph Morph;	Aktuelles Morphem	0
tMorph MorphInit;	Leere Struktur Morphem zur Initialisierung	0
char vBuf[1024];	Puffer zur Morphembildung	leerer String
char * pBuf;	Zeiger in den Puffer vBuf	vBuf
int line, col;	Morphemoposition in Zeile und Spalte	
	ZeichenklassenVektor	
	Automatentabelle	
	Funktionenarray	

Funktionen des Lexers

```
int initLex(char* fileName);
tMorph* lex();

Oder:
int lexInit(char* fileName);
tMorph* lexGetMorph();   // akt. Morphem
tMorph* lexNextMorph(); // synonym zu lex
```

Umsetzung der Automatentabelle in Beispielimplementierung

- Jeder Eintrag der Automatentabelle besteht hier aus einem Byte.
- Das niederwertige Halbbyte (4bit) enthält den Folgezustand
- Das höherwertige Halbbyte (4bit) enthält einen Funktionsindex, multipliziert mit 16, oder um 4 bit nach links verschoben

```
/* Zeichenklassenvector */
static char vZKl[128]=
/*      0  1  2  3  4 ...  D  E  F     */
/*--------------------..--------------*/
 /* 0*/{7, 7, 7, 7, 7,..., 7, 7, 7,/* 0*/
 /*10*/ 7, 7, 7, 7, 7,..., 7, 7, 7,/*10*/
 /*20*/ 7, 0, 0, 0, 0,..., 0, 0, 0,/*20*/
 /*30*/ 1, 1, 1, 1, 1,..., 4, 6, 0,/*30*/
 /*40*/ 0, 2, 2, 2, 2,..., 2, 2, 2,/*40*/
 /*50*/ 2, 2, 2, 2, 2,..., 0, 0, 0,/*50*/
 /*60*/ 0, 2, 2, 2, 2,..., 2, 2, 2,/*60*/
 /*70*/ 2, 2, 2, 2, 2,..., 0, 0, 0}/*70*/;

/* Schalt- und Ausgabefunktionen des Automaten */
static void fl   (void); // lesen
static void fb   (void); // beenden
static void fgl (void); // schreiben als Großbuchstaben, lesen
static void fsl (void); // schreiben, lesen
static void fslb(void); // schreiben, lesen, beenden

typedef void (*FX)(void);

static FX vfx[]={fl,fb,fgl,fsl,fslb}; // Vector von Pointern auf Funktion
```

Details der Beispielimplementierung

```
/* Funktionsindex *0x10, bzw. *16
typedef enum T_Fx
    {ifl=0x0,ifb=0x10,ifgl=0x20,ifsl=0x30,ifslb=0x40}tFx;

/* Morphemcodes */
typedef enum T_MC{mcEmpty, mcSymb, mcNum, mcIdent}tMC;

typedef enum T_ZS
  {
  zNIL,
  zERG=128,zLE,zGE,
  zBGN,zCLL,zCST,zDO,zEND,zIF,zODD,zPRC,zTHN,zVAR,zWHL
  }tZS;
```

```
static char vSMatrix[][8]=
/*        So      Zi   ...   '<'      '>'   Space
/*----------------------..------------------------*/
/* 0 */{0+ifslb,1+ifsl ...,4+ifsl ,5+ifsl ,0+ifl ,
/* 1 */ 0+ifb  ,1+ifsl ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 2 */ 0+ifb  ,2+ifsl ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 3 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 4 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 5 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 6 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 7 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb ,
/* 8 */ 0+ifb  ,0+ifb  ...,0+ifb  ,0+ifb  ,0+ifb
};

Finalzustand festhalten
static char vSMatrix[][]
/*        So      Zi   ...   '<'      '>'   Space
/*----------------------..------------------------*/
/* 0 */{0+ifslb,1+ifsl ...,4+ifsl ,5+ifsl ,0+ifl ,
/* 1 */ 1+ifb  ,1+ifsl ...,1+ifb  ,1+ifb  ,1+ifb ,
/* 2 */ 2+ifb  ,2+ifsl ...,2+ifb  ,2+ifb  ,2+ifb ,
. . .
/* 8 */ 8+ifb  ,8+ifb  ...,8+ifb  ,8+ifb  ,8+ifb
};
```

Automatentabelle in zwei Versionen.
Oben:
Wechsel auf Zustand 0 bei Beenden
Unten:
Finalzustand belassen.

Beide Varianten funktionieren, da der Lexer bei jedem neuen Aufruf im Zustand 0 startet.

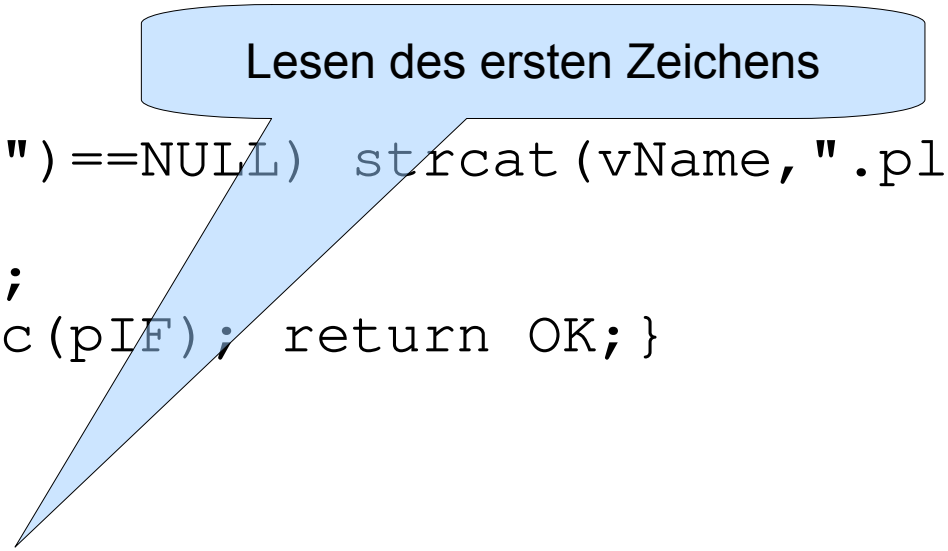
Eine dritte Variante wird auf der nachfolgenden Seite gezeigt.

Durch Einführen eines weiteren Zustandes (9) kann das Beenden des Automaten über die Automatentabelle gesteuert werden. Dabei wird keine neue Zeile für diesen Zustand benötigt, solange dieser Zustand den höchsten Index zugeordnet bekommt.

```
static char vSMatrix[][8]=
/*         So      Zi   ...    '<'     '>'   Space
/*---------------------...-----------------------*/
/* 0 */{9+ifslb,1+ifsl ... ,4+ifsl ,5+ifsl ,0+ifl ,
/* 1 */ 9+ifb  ,1+ifsl ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 2 */ 9+ifb  ,2+ifsl ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 3 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 4 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 5 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 6 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 7 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb ,
/* 8 */ 9+ifb  ,9+ifb  ... ,9+ifb  ,9+ifb  ,9+ifb
};
```

```
// Die Morphemstruktur

typedef struct
{
  tMC  MC;       /* Morphemcode */
  int  PosLine; /* Zeile       */
  int  PosCol;  /* Spalte      */
  union VAL
  {
    long Num;
    char*pStr;
    int  Symb;
  }Val;
  int  MLen;     /* Morphemlnge*/
}tMorph;


int initLex(char* fname);  // Initialisierung des Lexers
tMorph* Lex(void);         // Aufruf des Lexers
```

```
static FILE  *pIF;
static tMorph MorphInit; // alles mit 0 belegt
extern tMorph Morph;     // globale Variable f. Akt. Token
static int    X;         // Eingabezeichen
static int    Z;         // Aktueller Zustand
static char   vBuf[1024+1];    // Buffer zum Sammeln
static char*  pBuf;      // Pointer in den Buffer
static int    line,col;  // Zeile und Spalte
static int    End;       // Entfällt, wenn Zustand 9 → Ende

/*---- Initialisierung der lexiaklischen Analyse ----*/
int initLex(char* fname)
{
  char vName[128+1];

  strcpy(vName,fname);
  if (strstr(vName,".pl0")==NULL) strcat(vName,".pl0");

  pIF=fopen(vName,"r+t");
  if (pIF!=NULL) {X=fgetc(pIF); return OK;}
  return FAIL;
}
```



Algorithmus des Lexers

```
tMorph* Lex(void)
{
  Z=0;     // Anfangszustand
  int zx;  // Zustand merken
  Morph=MorphInit; // Morphem mit 0 löschen
  Morph.PosLine=line;
  Morph.PosCol =col;
  pBuf=vBuf; // Pointer auf Bufferanfang
  End=0;
  do
  {
    zx=  vSMatrix[Z][vZKl[X&0x7f]]&0xF;
    vfx[(vSMatrix[Z][vZKl[X&0x7f]])>>4]();
    Z=zx;
  }while (End==0); // (Z!=zEnd)  // zEnd:9
  return &Morph;
}
```

Folgezustand ermitteln

Funktion ausführen

Folgezustand einstellen

Das Verlassen der Schleife Wird hier über die Variable End von der Funktion Beenden Gesteuert, oder über Zustand Z9(Kommentar).

Die Funktionen

```
/*---- lesen ----*/
static void fl  (void)
{
  X=fgetc(pIF);
  if (X=='\n') line++,col=0;
  else col++;
}
/*---- schreiben als Grossbuchstabe, lesen ----*/
static void fgl (void)
{
  *pBuf=(char)toupper(X);// oder  *Buf=(char)X&0xdf;
  *(++pBuf)=0;
  fl();
}
/*---- schreiben, lesen ----*/
static void fsl (void)
{
  *pBuf=(char)X;
  *(++pBuf)=0;
  fl();
}
/*---- schreiben, lesen, beenden ----*/
static void fslb(void)
{
  fsl();fb();
}
```

Die Funktion Beenden

```
static void fb  ()
{
  int i,j;
  switch (Z)
  {
    /* Symbol */
    case 3: // :
    case 4: // <
    case 5: // >
    case 0:Morph.Val.Symb=vBuf[0];
      Morph.MC =mcSymb;
      break;
    /* Zahl */
    case 1:Morph.Val.Num=atol(vBuf);
      Morph.MC =mcNum;
      break;



  }
  Ende=1;  // entfällt bei Variante mit Zustand zEnd
}
```

Weitere Casezweige

```
/* Ergibtzeichen */
case 6:Morph.Val.Symb=(long)zErg;
       Morph.MC =mcSymb;
       break;
/* KleinerGleich */
case 7:Morph.Val.Symb=(long)zle;
       Morph.MC =mcSymb;
       break;
/* GroesserGleich */
case 8:Morph.Val.Symb=(long)zge;
       Morph.MC =mcSymb;
       break;
```

Die zusammengesetzten Sonderzeichen erhalten als Morpheminhalt einen Code, der im Headerfile wie folgt vereinbart ist:

```
typedef enum T_ZS
{
zNIL,
zErg=128,zle,zge,
. . .
}
```

Schlüsselworterkennung

- Wird für alle erkannten Zeichenfolgen, die mit einem Buchstaben beginnen, durchlaufen (sehr oft)
- Intelligente Lösungen bestimmen hier das Laufzeitverhalten des Compilers wesentlich mit.
- Nachfolgend werden verschiedene Möglichkeiten der Schlüsselworterkennung aufgeführt. Das Ziel ist, eine möglichst schnelle Erkennung von Schlüsselworten.
- Am schnellsten ist wahrscheinlich, wenn die Schlüsselworte direkt durch den Automaten erkannt werden (Variante 5), jedoch ist das sehr aufwendig.
- Ist ein Schlüsselwort erkannt, wird daraus ein Morphem vom Typ Symbol und das Morphem bekommt einen Schlüsselwortcode als Wert (in einem enum vereinbart).

```
typedef enum T_ZS
  {
  zNIL,
  zErg=128,zle,zge,
  zBGN,zCLL,zCST,zDO,zEND,zELS,zIF,zGET,zODD,zPUT,zPRC,zTHN,zVAR,zWHL
  //zBGN,zCLL,zCST,zDO,zELS,zEND,zIF,zODD,zPRC,zTHN,zVAR,zWHL
  }tZS;
```

Schlüsselwortekennung

Verschiedene Realisierungsmöglichkeiten:

1. Das Einstiegsmodell:

   Schlüsselwörter, in einem Vektor gespeichert, durchsuchen.

2. Schlüsselworte liegen sortiert vor, Abbruch, wenn gesuchtes Wort kleiner als zu vergleichendes Schlüsselwort ist.

3. Binäre Suche in sortiertem Vektor oder Suche in Baum

4. Hashtechniken

Einbeziehung der Automatentabelle

5. Könnte auch bereits durch die Automatentabelle gelöst werden, jedoch hat dann die Zeichenklassifizierung kaum noch Sinn. Für jedes Schlüsselwort werden so viele Zustände benötigt, wie es Zeichen hat, für jedes Zeichen, das in einem Schlüsselwort vorkommt, wird eine Spalte benötigt.
6. Finalzustand „Potenzielles Schlüsselwort“ wird gebildet, wenn ein Token nur aus Zeichen, die in Schlüsselworten vorkommen, besteht (Neue Zeichenklasse erforderlich).
7. Um die Schlüsselworterkennung nur aufzurufen, wenn sie wirklich benötigt wird, könnte man weitere Zustände einführen. Ein Schlüsselwort ist mindestens 2 Zeichen lang und enthält keine Ziffern (keine neue Zeichenklasse erforderlich)
8. Neue Zeichenklasse für alle Buchstaben, mit denen ein Schlüsselwort beginnt.

Verschiedene Implementationen

- Binäre Suche
- Arbeit mir der Automatentabelle
- Hashtechnik

Binäre Suche

```
int main(int argc, char*argv[])
{
  const char*  Keyw[]= /* sortierter Vektor der Keywords */
            {
    "BEGIN","CALL","CONST","DO","END","IF","ODD",
    "PROCEDURE","THEN","VAR","WHILE"
            };

  int n=sizeof Keyw/sizeof(char*);
  int i;
  printf("Anz:%d\n",n);

  i=binary_search(Keyw,n+1,argv[1]);

  if (i>=0)printf("Ergebnis: %d %s\n",i, Keyw[i]);
  else     printf("Ergebnis: not found\n");
  return 0;
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int binary_search(» const char** M,»// hey stack
»    »      » »    »    int n,» »    »    // number
»    »      » »    »    const char* X)» // needle
{
    unsigned mitte;
    unsigned links = 0; /* man beginne beim kleinsten Index */
    unsigned rechts = n - 1; /* Arrays sind 0-basiert */
    int ret=-1;
    int bool;

    do
    {
        if (n<=0) break;
        mitte = links + ((rechts - links) / 2); // Bereich halbieren

        if (rechts < links) break; // alles wurde durchsucht, X nicht gefunden
»
        bool=strcmp(M[mitte],X);
»
        if (bool==0) ret=mitte;      else  // Element X gefunden?
        if (bool >0) rechts = mitte; else  // im linken Abschnitt weitersuchen
                     links = mitte + 1;    // im rechten Abschnitt weitersuchen
        n=(n)/2;
    }while (bool!=0);
    return ret;
}
```

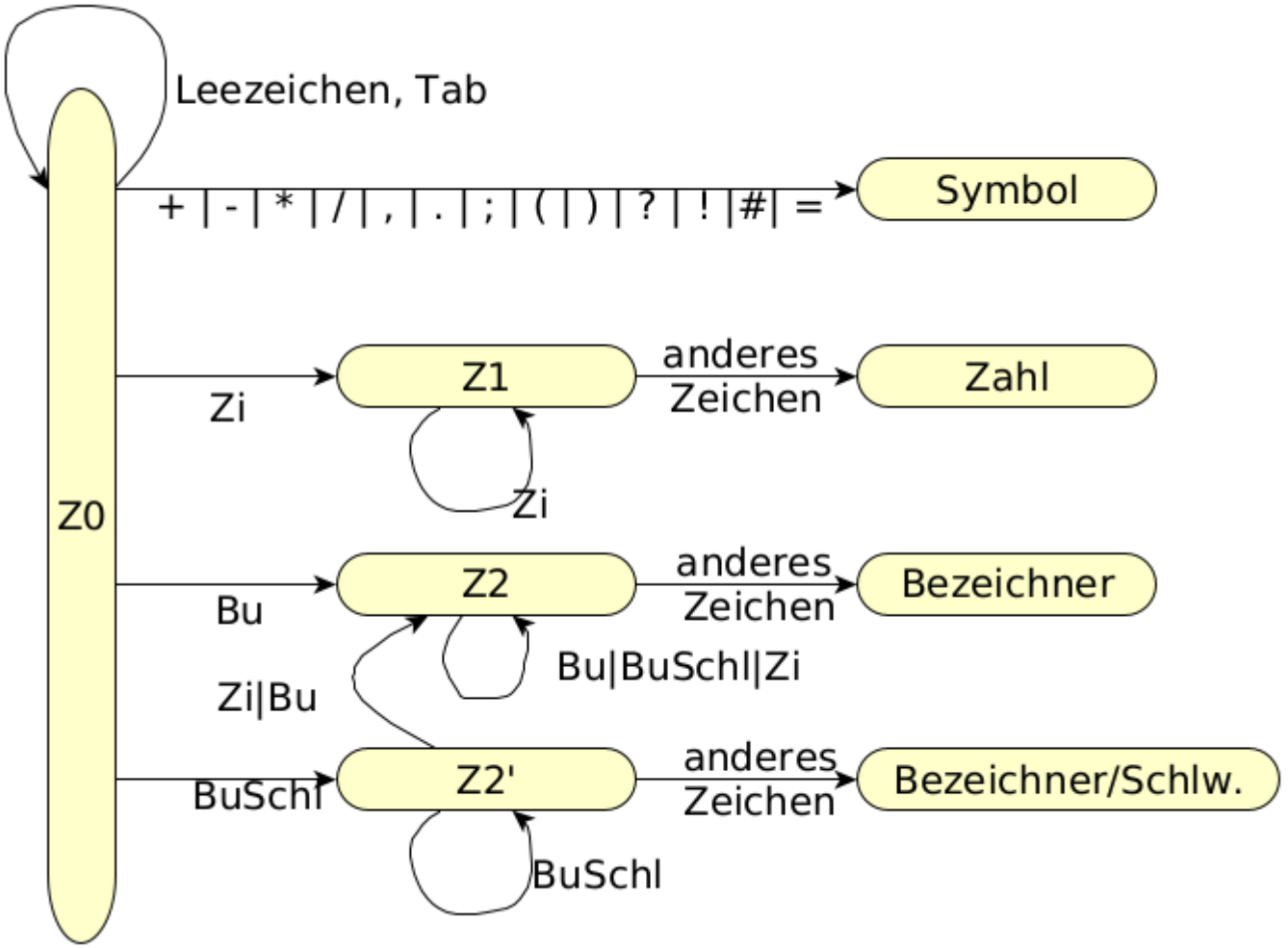
Rekursive Variante

```
int binary_search(» const char** M, // stock
»    »      » »    »   const char* X,» // needle
»    »      » »    »   unsigned int   left,
                       unsigned int right)»
{
  unsigned middle, index, ret=-1;

  if (M == NULL) // Bereichsüberprüfung
    printf("out of Range\n");
  else
  {
    int bool;
    int diff= (((right) - (left)) / 2); // Bereich halbieren
    middle = left + diff;

    if (right >= left && diff>=0)
    {
      bool=strcmp(M[middle],X);
      if (bool==0) ret= middle;       // gefunden
      else
      {
          if (bool >0)right=middle;
          else        left =middle+1;
          if (left!=right /*&& diff>0*/)
            ret= binary_search(M, X, left, right);
      }
    }
  }
  return ret;
}
```

Übersicht von Zeichen, die in Schlüsselworten vorkommen

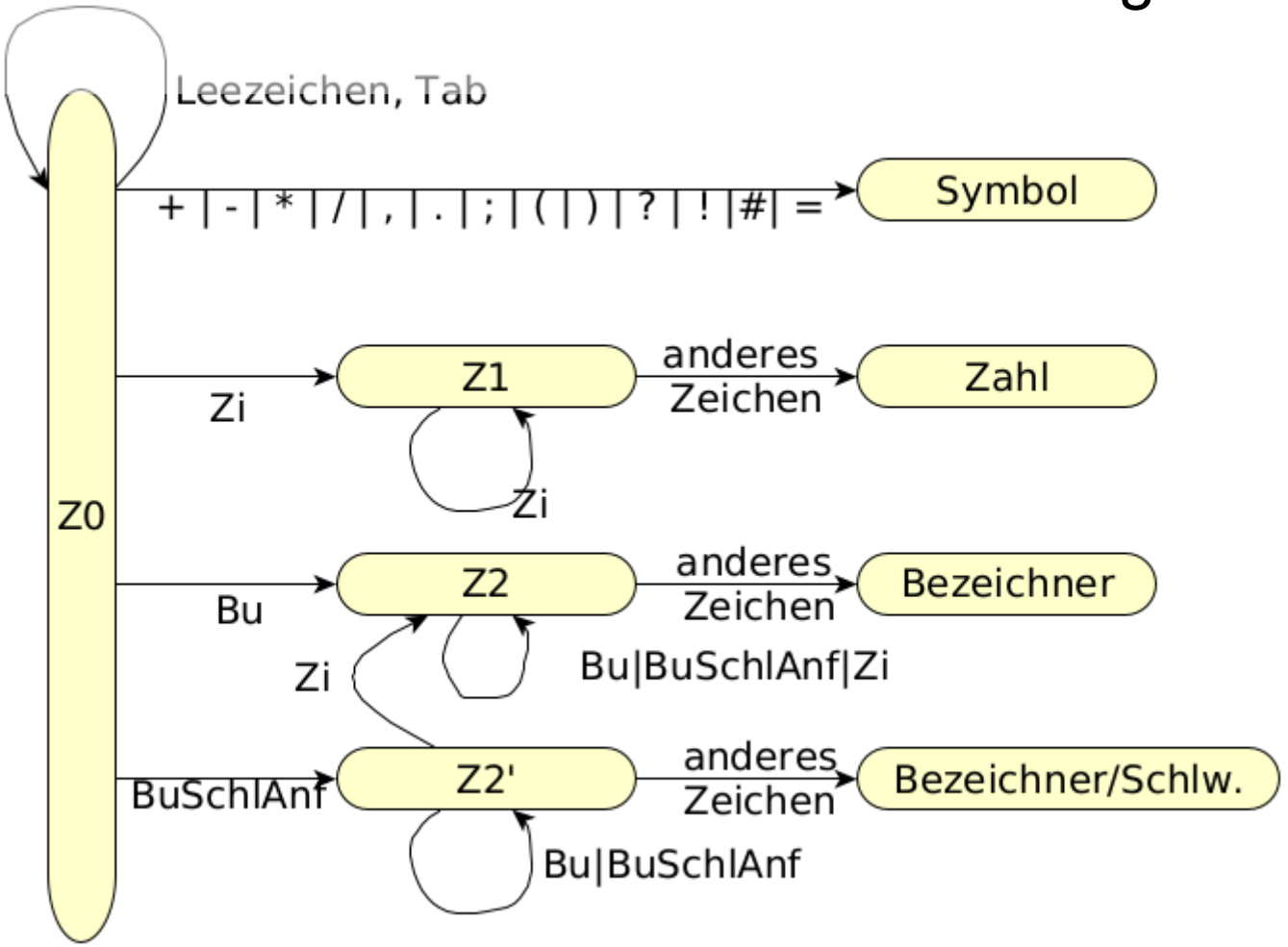
	kommt vor	Beginnt mit		begin	call	const	do	end	if	odd	procedure	then	var	while
A	X				x								x	
B	X	X		x										
C	X	X			x	x					x			
D	X	X					x	x		x	x			
E	X	X		x				x			x	x		x
F	X								x					
G	X			x										
H	X											x		x
I	X	X		x					x					x
J														
K														
L	X				x									x
M														
N	X			x		x		x				x		
O	X	X				x	x			x	x			
P	X	X									x			
Q														
R	X										x		x	
S	X					x								
T	X	X				x						x		
U	X										x			
V	X	X											x	
W	X	X												x
X														
Y														
Z														

Zeichenklasse für Buchstaben der Schlüsselworte Variante 6/ 7



Endet der Automat im Zustand Schlüsselwort/Bezeichner, muss geprüft werden, ob es ein Schlüsselwort ist oder ein Bezeichner, jedoch wird das seltener geschehen, als in der Ausgangsvariante

Zeichenklasse für Buchstaben der Schlüsselwortanfänge (Variante 8)



Es gibt eine Zeichenklasse für Buchstaben am Schlüsselwortanfang. Liegt ein solcher Buchstabe in Zustand 0 vor, so wird in Zustand Z2‘ gewechselt.
Folgen nun Buchstaben, so kann es sich um ein Schlüsselwort handeln. Endet der Automat im Zustand Schlüsselwort/Bezeichner, muss geprüft werden, ob es ein Schlüsselwort ist oder ein Bezeichner, jedoch wird das seltener geschehen, als in der Ausgangsvariante.
Endet des Automat in Z2 liegt kein Schlüsselwort vor.

Automatentabelle mit Schlüsselwörtern BEGIN, CALL, CONST Variante 5

		So	Zi	Bu	:	=	<	>	sonst	A	B	C	D	E	F	G	H	I	L	N	O	P	R	S	T	U	V	W
sozei	0	0slb	1sl	2gl	3sl	0slb	4sl	5sl	0l	2gl	**9gl**	**14gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
Zahl	1	1b	1sl	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b	1b
ident	2	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
:	3	3b	3b	3b	3b	6sl	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b	3b
<	4	4b	4b	4b	4b	7sl	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b	4b
>	5	5b	5b	5b	5b	8sl	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b	5b
:=	6	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b	6b
<=	7	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b	7b
>=	8	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b	8b
	9	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	**10gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	10	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	**11gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	11	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**12gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	12	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**13gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
BEGIN	13	**13b**	2sl	2gl	**13b**	**13b**	**13b**	**13b**	**13b**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	14	2b	2sl	2gl	2b	2b	2b	2b	2b	**15gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**18gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	15	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**16gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	16	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**17gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
CALL	**17**	**17b**	2sl	2gl	**17b**	**17b**	**17b**	**17b**	**17b**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	18	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**19gl**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl
	19	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**20gl**	2gl	2gl	2gl	2gl
	20	2b	2sl	2gl	2b	2b	2b	2b	2b	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	**21gl**	2gl	2gl	2gl
CONST	21	**21b**	2sl	2gl	**21b**	**21b**	**21b**	**21b**	**21b**	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl	2gl

Es ist unschwer zu erkennen, dass die Automatentabelle auf diese Weise sehr groß und unübersichtlich wird. Im Beispiel sind lediglich 3 Schlüsselwörter berücksichtigt.

Schlüsselworttabelle mit einfacher Hashtechnik (statische Hashtable)

Die Hashfunktion berechnet aus Anfangsbuchstaben eines erkannten Bezeichners und Länge -2 die Stelle in der Tabelle. Wird eine 0 gefunden, ist es kein Schlüsselwort, ansonsten entscheidet der Stringvergleich ab dem 2. Zeichen.Vorteilhaft ist, dass es keine Dopplungen gibt – sehr schnell, nachteileig ist die sehr dünne Besetzung der Matrix.

```
tKeyWordTab mSclW['Z'-'A'+1][8]=
{
/*Len:     2           3           4           5           6           7           8           9     *
/* A */ {{0L ,zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* B */ {{0L ,zNIL},{0L,  zNIL},{0L,   zNIL},{"EGIN",zBGN},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* C */ {{0L ,zNIL},{0L,  zNIL},{"ALL",zCLL},{"ONST",zCST},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* D */ {{"O",zDO },{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* E */ {{0L ,zNIL},{"ND",zEND},{"LSE",zELS},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* F */ {{0L ,zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* G */ {{0L ,zNIL},{"ET",zGET},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* H */ {{0L ,zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* I */ {{"F",zIF },{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* J */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* K */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* L */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* M */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* N */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* O */ {{0L, zNIL},{"DD",zODD},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* P */ {{0L, zNIL},{"UT",zPUT},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{"ROCEDURE",zPRC}},
/* Q */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* R */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* S */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* T */ {{0L, zNIL},{0L,  zNIL},{"HEN",zTHN},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* U */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* V */ {{0L, zNIL},{"AR",zVAR},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* W */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{"HILE",zWHL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* X */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* Y */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}},
/* Z */ {{0L, zNIL},{0L,  zNIL},{0L,   zNIL},{ 0L,   zNIL},{0L,zNIL},{0L,zNIL},{0L,zNIL},{0L,        zNIL}}};
```

Der Code dazu aus der Funktion beenden.

```
    /* Bezeichner/Wortsymbol */
    case 2:i=vBuf[0]-'A';
           j=strlen(vBuf)-2;
           if (j>=0)
             if (mSclW[i][j].pKeyWord)
               if (strcmp(vBuf+1,mSclW[i][j].pKeyWord)==0)
               {
                 Morph.MC =mcSymb;
                 Morph.Val.Symb=mSclW[i][j].KWCode;
»      »      break;
               }
           Morph.Val.pStr=vBuf;
           Morph.MC =mcIdent;
           break;
```

```
typedef enum T_ZS
  {
  zNIL,
  zErg=128,zle,zge,
  zBGN,zCLL,zCST,zDO,zEND,zELS,zIF,zGET,zODD,zPUT,zPRC,zTHN,zVAR,zWHL
  //zBGN,zCLL,zCST,zDO,zELS,zEND,zIF,zODD,zPRC,zTHN,zVAR,zWHL
  }tZS;
```