[File: BNF.WPW]            <Erster Selbstbezug>

# DER COMPILER-COMPILER BNF.

<ZweiterSelbstbezug>

## Entwicklung eines Compilers für beliebige formale Sprachen.

Wolf-Werner Scheuermann, Hamburg 1993

# Compilerbau

Ein Compiler ist ein Programm zur Übersetzung von Programmen, die in einer bestimmten formalen Sprache geschrieben sind, in eine für den Computer verständliche Form: die Maschinensprache.

Normalerweise ist das eine geradlinige Sache: hier der Quelltext, dort das Binärformat. Und dazwischen der Compiler. Allerdings versteht dieser Compiler genau eine formale Sprache. An dieser Stelle kommt also die Frage auf, ob es möglich ist einen Compiler für beliebige formale Sprachen zu bauen, also einen Compiler, der die Grammatik einer beliebigen Sprache liest und dann in der Lage ist, Sätze, Texte, Programme in dieser Sprache zu verstehen und zu übersetzen. Im folgenden wird gezeigt, wie diese Aufgabe zu lösen ist und welche Einschränkungen gemacht werden müssen, aber auch, welche Möglichkeiten sich öffnen.

Der erste Schritt besteht in der korrekten Formulierung einer Sprache zur Beschreibung der Syntax von formalen Sprachen. Zweckmäßigerweise wählt man dafür die Backus-Naur-Form, welche in ASCII-Form leicht als String verarbeitet werden kann. Diese Form gewährleistet darüberhinaus die Kontextfreiheit der damit definierten formalen Sprache.

Backus-Naur-Form:

---

```
BNF=Words`='Expression`.'.
Expression=|Term|{`|'Term}`!'!.
Term=Factor{Factor}.
Factor=|Words|`Symbol'Symbol|``'Symbol`''|`['Expression`]'|`{'Expression`}'!.@
==============
```

Dieses ist die Beschreibung der Backus-Naur-Form mit Hilfe der Backus-Naur-Form, denn diese ist selbst eine formale Sprache.
<Dritter Selbstbezug>

Vier Erläuterungen zu dieser Metasprache sind notwendig:

1)Klammerungen jeglicher Art (also z.B. ` und ', oder | und ! oder [ und ]) müssen aus unterschiedlichen Anfangs- und Endezeichen bestehen.

2) Zum Lesen muß ein Ende-Zeichen nach der letzten Zeile angehängt werden (hier ein Klammeraffe): ... ' | ` { ' Expression ` } ' ! . @

3) Für die erste Version muß zwischen jedes Symbol ein Leerzeichen (Blank) eingefügt werden:

Term = Factor { Factor } .
statt:
Term=Factor{Factor}.

(Das erlaubt fürs erste eine simple Scanner-Version. Ein Scanner liest den Text und separiert einzelne Wörter.).

4) Das erste Wort der Syntax (hier "BNF") dient als Name der Sprache und der Haupt-Subroutine sowie als Filename.
<Vierter Selbstbezug>

5) Es gibt zwei durch BNF nicht definierte Wörter: "Symbol" und "Words". "Symbol" ist gewissermaßen der archimedische Punkt der ganzen Sprachanalyse. Die Bedeutung (Semantik) dieses Wortes - und die ist hier identisch mit seiner Funktion - ist: "Lies und verstehe ein Symbol". "Words" ist ein (Meta-)Metasymbol, welches für die Benennung der einzelnen Regeln der Sprache steht.

```
==============
[File: BNF.SYN]

BNF = Words ` = ' Expression ` . ' .
Expression = | Term | { ` | ' Term } ` ! ' ! .
Term = Factor { Factor } .
Factor = | Words | ` Symbol ' Symbol | ` ` ' Symbol ` ' ' | ` [ ' Expression `] ' | ` { '
Expression ` } ' ! . @
==============
```

Die Übersetzung der Metasprache (Syntax-Beschreibungssprache) in Programmstrukturen geschieht nach folgenden Regeln. Als Programmiersprache wurde Microsofts Q-BASIC bzw. Quick-BASIC verwendet. Großbuchstaben bezeichnen Worte, Kleinbuchstaben Symbole. Auch die Umsetzung der Backus-Naur-Form in Syntax-Graphen ist gezeigt:

---

```
A=... .

DECLARE SUB A ()                                A-->...

SUB A
  .
  .                                             ...-->[ A ]-->...
  .
END SUB
```

---

```
|B|C|...|D!

  IF ch IN first(B) THEN                        ...-+->[ B ]--+->...
    B                                               |         ^
  ELSEIF ch IN first(C) THEN                        v         |
    C                                               +->[ C ]->+
          ...                                       |         |
  ELSEIF ch IN first(D) THEN                        .         .
    D                                               .         .
  ELSE                                              |         ^
    ERR0R                                           +->[ D ]->+
  END IF
```

---

```
AB...C

  A
  B                                             ...->[ A ]->[ B ]...[ C ]->...
  ...
  C
```

---

```
{A}

  WHILE ch IN first(A)                          ...-+---------+->...
    A                                               ^         |
  WEND                                              +<-[ A ]<-+
```

---

```
{aA}

  WHILE ch = "a"                                ...-+--------------+->...
    getword                                         ^              |
    A                                               +<-[ A ]<-(a)<-+
  WEND
```

---

```
[A]

  IF ch IN first(A) THEN                        ...-+---------+->...
    A                                               |         ^
  END IF                                            +->[ A ]->+
```

---

```
[aA]

  IF ch = "a" THEN                                    ...-+--------------+->...
    getword                                               |              ^
    A                                                     + ->(a)->[ A ]->+
  END IF
```

_____________

```
a                                                     ...->-(a)->...

  IF ch = "a" THEN
    getword
  ELSE
    ERR0R
  END IF
```

_____________

```
|aA|bB|...|cC!

  IF ch = "a" THEN
    getword
    A
  ELSEIF ch = "b" THEN
    B
        ...
  ELSEIF ch = "c" THEN
    C
  ELSE
    ERR0R
  END IF
```

==============

Allgemeines zu den Syntaxgraphen:
--------------------------------
Regel 1) Keine zwei Äste innerhalb eines Graphen dürfen mit dem gleichen Symbol beginnen.

Regel 2) Es muß eindeutig entscheidbar sein, ob das nächste Symbol zu dem Graphen oder bereits zu seinem Nachfolger gehört. Die Mengen der Anfangssymbole müssen daher disjunkt sein. Für jedes nichtterminale Symbol A wird die zugehörige Menge first(A) der Anfangssymbole bestimmt.

Wendet man die Übersetzungsregeln nun von Hand auf die Beschreibung der Backus-Naur-Form an, so erhält man die Subroutinen des folgenden Programms. Das Hauptprogramm ergibt sich mehr oder weniger zwangsläufig und wird in der weiteren Entwicklung unverändert übernommen.

Dieses Programm (der Translator) erzeugt aus einer Syntax einen Parser (ein weiteres Programm), der in der Lage ist, Sätze in der durch die Syntax definierten Sprache auf syntaktische Korrektheit zu überprüfen. Der Translator selbst führt noch keinen Syntaxcheck durch.

Der Translator erinnert an eine Turing-Maschine: Er liest Zeichen ein und ändert daraufhin in Abhängigkeit von seinem aktuellen Zustand eben denselben und gibt Zeichen aus.

```
==============
[File: TRANSLAT.BAS]

'BNF-Translator   (C) by W.-W.Scheuermann  07-24-1993

DECLARE SUB Translation ()

DECLARE SUB BEGIN ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DIM SHARED sentence AS STRING
DIM SHARED nextch AS STRING
DIM SHARED ch AS STRING
DIM SHARED lastch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING
DIM SHARED event AS STRING

CONST COMMAND = "BNF"          'Q-BASIC Version only!

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1
CONST outfile = 2

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    WHILE sentence <> ""
      WHILE ch = ""
        getword
      WEND
      Translation
    WEND
  WEND
FIN

SUB BEGIN
  CLS
  file = COMMAND
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
  OPEN path + file + ".PRS" FOR OUTPUT AS outfile
  event = "end"
  PRINT #outfile, "'"; COMMAND; "-Parser  (C) by W.-W.Scheuermann  ";
  PRINT #outfile, DATE$
  PRINT #outfile, "'Nicht lauffähig!"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE SUB Symbol ()"
  PRINT #outfile, "DECLARE SUB Words ()"
  PRINT #outfile, "DECLARE SUB BEGIN ()"
  PRINT #outfile, "DECLARE SUB ERR0R ()"
  PRINT #outfile, "DECLARE SUB FIN ()"
  PRINT #outfile, "DECLARE SUB getsentence ()"
  PRINT #outfile, "DECLARE SUB getword ()"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE FUNCTION chINfirst(char AS STRING)"
  PRINT #outfile, ""
  PRINT #outfile, "DIM SHARED sentence AS STRING"
  PRINT #outfile, "DIM SHARED lastch AS STRING"
  PRINT #outfile, "DIM SHARED ch AS STRING"
  PRINT #outfile, "DIM SHARED nextch AS STRING"
  PRINT #outfile, "DIM SHARED file AS STRING * 8"
  PRINT #outfile, "DIM SHARED path AS STRING"
  PRINT #outfile, ""
  PRINT #outfile, "CONST FALSE = 0"
  PRINT #outfile, "CONST TRUE = NOT FALSE"
  PRINT #outfile, "CONST syntaxfile = 1"
  PRINT #outfile, ""
  PRINT #outfile, "CONST COMMAND = "; CHR$(34); "BNF"; CHR$(34); "          "; "'"; "Q-BASIC
Version only!"
  PRINT #outfile, ""
  PRINT #outfile, "BEGIN"
  PRINT #outfile, "  WHILE NOT EOF(syntaxfile)"
  PRINT #outfile, "    getsentence"
  PRINT #outfile, "    getword"
  PRINT #outfile, "    IF NOT EOF(syntaxfile) THEN"
  PRINT #outfile, "      WHILE ch = "; CHR$(34); CHR$(34)
  PRINT #outfile, "        getword"
  PRINT #outfile, "      WEND"
  PRINT #outfile, "      "; COMMAND
  PRINT #outfile, "    END IF"
  PRINT #outfile, "  WEND"
  PRINT #outfile, "FIN"
  PRINT #outfile, ""
  PRINT #outfile, "SUB BEGIN"
  PRINT #outfile, "  CLS"
  PRINT #outfile, "  file = COMMAND'$"
  PRINT #outfile, "  path = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  OPEN path + file + "; CHR$(34); ".SYN"; CHR$(34); " FOR INPUT AS syntaxfile"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Symbol"
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB ERR0R"
  PRINT #outfile, "  PRINT "; CHR$(34); "      "; CHR$(34); "; CHR$(27); "; CHR$(34); "SYNTAX
ERROR!"; CHR$(34)
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB FIN"
  PRINT #outfile, "  CLOSE #syntaxfile"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getsentence"
  PRINT #outfile, "  LINE INPUT #syntaxfile, sentence"
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getword"
  PRINT #outfile, "  lastch = ch"
  PRINT #outfile, "  ch = nextch"
  PRINT #outfile, "  IF INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") = 0 THEN"
  PRINT #outfile, "    cha$ = sentence"
  PRINT #outfile, "    sentence = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  ELSE"
  PRINT #outfile, "    cha$ = LEFT$(sentence, INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") -
1)"
  PRINT #outfile, "    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, "; CHR$(34); "
"; CHR$(34); "))"
  PRINT #outfile, "  END IF"
  PRINT #outfile, "  nextch = cha$"
  PRINT #outfile, "  PRINT ch;"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Words"
  PRINT #outfile, " 'Here the names of the rules of the language are parsed."
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "FUNCTION chINfirst (char AS STRING)"
  PRINT #outfile, "  chINfirst = FALSE"
  PRINT #outfile, "  SELECT CASE char"
  PRINT #outfile, "    CASE "; CHR$(34); "WHILE"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "|"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Words"; CHR$(34); ""
  PRINT #outfile, "      IF NOT (ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34);
"!"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR
```

```
ch = "; CHR$(34); "'"; CHR$(34); " OR ch = "; CHR$(34); "[";
  PRINT #outfile, CHR$(34); " OR ch = "; CHR$(34); "]"; CHR$(34); " OR ch = "; CHR$(34); "{";
CHR$(34); " OR ch = "; CHR$(34); "}"; CHR$(34); " OR ch = "; CHR$(34); "."; CHR$(34); ") THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Expression"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "Words"; CHR$(34); " OR chINfirst("; CHR$(34);
"Words"; CHR$(34); ") OR ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$
(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = ";
  PRINT #outfile, CHR$(34); "["; CHR$(34); " OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Term"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Factor"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "etc."; CHR$(34); "  'Add user specific conditions"
  PRINT #outfile, "      IF ch = "; CHR$(34); "first('etc.')"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "  END SELECT"
  PRINT #outfile, "END FUNCTION"
  PRINT #outfile, ""
END SUB

SUB FIN
  CLOSE #syntaxfile
  CLOSE #outfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT cha$;
END SUB

SUB Translation
  getword
  IF lastch = "`" AND nextch = "'" THEN
    IF event = "level" THEN
      PRINT #outfile, "  IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "else" THEN
      PRINT #outfile, "IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "while" THEN
      PRINT #outfile, " ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    END IF
  ELSEIF ch = "!" AND nextch = "." THEN
    PRINT #outfile, "  ELSE"
    PRINT #outfile, "    ERR0R"
    PRINT #outfile, "  END IF"
  ELSEIF lastch = "|" AND ch = "{" AND nextch = "`" THEN
    PRINT #outfile, "IF chINfirst("; CHR$(34); "WHILE"; CHR$(34); ") THEN"
    PRINT #outfile, "    WHILE";
    event = "while"
  ELSEIF ch = "{" THEN
    PRINT #outfile, "  WHILE";
    event = "while"
  ELSEIF ch = "}" THEN
    PRINT #outfile, "  WEND"
  ELSEIF ch = "|" AND lastch = "=" THEN
    PRINT #outfile, "  ";
    event = "else"
  ELSEIF ch = "|" THEN
    PRINT #outfile, "  ELSE";
    event = "else"
  ELSEIF ch = "[" AND nextch = "`" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); "IF"; CHR$(34); ") THEN"
  ELSEIF ch = "[" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); nextch; CHR$(34); ") THEN"
  ELSEIF ch = "]" THEN
    PRINT #outfile, "  END IF"
  ELSEIF ch = "`" THEN
  ELSEIF ch = "'" THEN
    IF event = "else" THEN
      PRINT #outfile, " THEN"
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "while" THEN
      PRINT #outfile, ""
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "level" THEN
      PRINT #outfile, " THEN"
      PRINT #outfile, "    getword"
      PRINT #outfile, "  ELSE"
      PRINT #outfile, "    ERR0R"
      PRINT #outfile, "  ENDIF"
    END IF
    event = "level"
  ELSEIF ch = "=" THEN
    PRINT #outfile, "DECLARE SUB "; lastch; " ()"
    PRINT #outfile, ""
    PRINT #outfile, "SUB "; lastch
    event = "level"
  ELSEIF ch = "." THEN
    PRINT #outfile, "END SUB"
    PRINT #outfile, ""
    event = "end"
  ELSE
    IF event = "level" THEN
      PRINT #outfile, "  "; ch
    ELSEIF event = "while" THEN
      PRINT #outfile, " chINfirst("; CHR$(34); ch; CHR$(34); ") "
      PRINT #outfile, "    "; ch
    ELSEIF event = "else" THEN
      PRINT #outfile, "  IF chINfirst("; CHR$(34); ch; CHR$(34); ") THEN"
      PRINT #outfile, "    "; ch
    END IF
  END IF
END SUB
==============
```

Wendet man den Translator auf die Syntax der Backus-Naur-Form an, so erhält man einen Parser, der eine Syntax auf Konformität mit BNF prüfen kann - also auch die Syntax der Backus-Naur-Form selbst. <Fünfter Selbstbezug>

Der folgende Text ist das vom Translator erzeugte Programm. Es ist in dieser Rohform noch nicht unter einem BASIC-Interpreter lauffähig. Die Bedingungen in der Funktion

```
  chINfirst(...)
```

müssen noch von Hand explizit formuliert werden. Auch muß die optische Strukturierung des Programms noch von Hand nachgearbeitet werden.

==============

```
[File: BNF.PRS]

'BNF-Parser  (C) by W.-W.Scheuermann  07-24-1993
'Nicht lauffähig!

DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst(char AS STRING)

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "BNF"          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN

SUB BEGIN
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
END SUB

SUB Symbol
  getword
END SUB
```

```
SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB FIN
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  getword
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

DECLARE SUB BNF ()

SUB BNF
  Words
  IF ch = "=" THEN
    getword
  ELSE
    ERR0R
  ENDIF
  Expression
  IF ch = "." THEN
    getword
  ELSE
    ERR0R
  ENDIF
END SUB

DECLARE SUB Expression ()

SUB Expression
    IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
    getword
  Term
  WEND
  IF ch = "!" THEN
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB
```

```
DECLARE SUB Term ()

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB

DECLARE SUB Factor ()

SUB Factor
    IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
  Symbol
  ELSEIF ch = "`" THEN
    getword
  Symbol
  IF ch = "'" THEN
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "[" THEN
    getword
  Expression
  IF ch = "]" THEN
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "{" THEN
    getword
  Expression
  IF ch = "}" THEN
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB
==============
```

Das Ergebnis diese Nachbearbeitungsprozesses stellt der folgende, nun lauffähige BNF-Parser dar. Mit seiner Hilfe läßt sich z.B. die Korrektheit der BNF-Syntax selbst sehr leicht überprüfen.

Als Ergebnis und Konsequenz dieser Überprüfung ergab sich z.B. die Erläuterung 1) zur BNF-Form.

Es ist zu beachten, daß das Programm, entsprechend der Stuktur der BNF hochgradig rekursiv arbeitet.
<Sechster Selbstbezug>

```
==============
[File: BNF.BAS]

'BNF-Parser  (C) by W.-W.Scheuermann  07-24-1993
'Lauffähig!

DECLARE SUB BNF ()
DECLARE SUB Expression ()
DECLARE SUB Term ()
DECLARE SUB Factor ()

DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst (char AS STRING)

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "BNF"          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN
```

```
SUB BEGIN
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
END SUB

SUB BNF
  Words
  IF ch = "=" THEN
    getword
  ELSE
    ERR0R
  END IF
  Expression
  IF ch = "." THEN
    getword
  ELSE
    ERR0R
  END IF
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB Expression
  IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
      getword
      Term
    WEND
    IF ch = "!" THEN
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB Factor
  IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
    Symbol
  ELSEIF ch = "`" THEN
    getword
    Symbol
    IF ch = "'" THEN
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "[" THEN
    getword
    Expression
    IF ch = "]" THEN
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "{" THEN
    getword
    Expression
    IF ch = "}" THEN
```

```
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB FIN
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Symbol
  getword
END SUB

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB


SUB Words
 'Here the names of the rules of the language are parsed.
  getword
END SUB
==================
```

Im nächsten Schritt wird nun der BNF-Parser dergestalt erweitert, daß er als Ausgabe wiederum einen BNF-Parser erzeugt, der seinerseits einen BNF-Parser generieren kann, wenn er auf die BNF-Syntax angewendet wird und so fort. <Siebenter Selbstbezug>

Das geschieht, indem den terminalen Symbolen der Backus-Naur-Form (z.B. { oder [, etc.) jeweils eine Subroutine zugeordnet wird, in der die Semantik des betreffenden Symbols behandelt wird. Es können dabei die Verfahren des Translators direkt verwendet werden. Der Translator hat sich damit auf einer höheren Stufe reproduziert. <Achter Selbstbezug>
Er wird zum BNF-Compiler.

Dieser BNF-Compiler erzeugt für beliebige Sprachdefinitionen einen Parser, der durch Ergänzungen des Benutzers zum Compiler bzw. Interpreter für die be-treffende Sprache wird. Der BNF-Compiler prüft dabei zusätzlich die Konformität der Sprachdefiniton mit BNF.

```
==============
[File: BNF1.BAS]

'BNF-Parser  (C) by W.-W.Scheuermann  07-24-1993
'Lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Semantic2 ()
DECLARE SUB Semantic3 ()
DECLARE SUB Semantic4 ()
DECLARE SUB Semantic5 ()
DECLARE SUB Semantic6 ()
DECLARE SUB Semantic7 ()
DECLARE SUB Semantic8 ()
DECLARE SUB Semantic9 ()
DECLARE SUB Semantic10 ()
DECLARE SUB Semantic11 ()

DECLARE SUB BNF ()
DECLARE SUB Expression ()
DECLARE SUB Term ()
DECLARE SUB Factor ()

DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst (char AS STRING)

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

DIM SHARED event AS STRING
DIM SHARED sym(1 TO 30) AS STRING
DIM SHARED Semantic AS INTEGER

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST outfile = 2

CONST COMMAND = "BNF"          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN

SUB BEGIN
  Semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
  OPEN path + file + ".CMP" FOR OUTPUT AS outfile
  event = "end"
  PRINT #outfile, "'"; COMMAND; "-Compiler  (C) by W.-W.Scheuermann  ";
  PRINT #outfile, DATE$
  PRINT #outfile, "'Nicht lauffähig!"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE SUB Semantic0 ()"
  PRINT #outfile, "DECLARE SUB Semantic1 ()"
  PRINT #outfile, "DECLARE SUB Symbol ()"
  PRINT #outfile, "DECLARE SUB Words ()"
  PRINT #outfile, "DECLARE SUB BEGIN ()"
  PRINT #outfile, "DECLARE SUB ERR0R ()"
  PRINT #outfile, "DECLARE SUB FIN ()"
  PRINT #outfile, "DECLARE SUB getsentence ()"
  PRINT #outfile, "DECLARE SUB getword ()"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE FUNCTION chINfirst(char AS STRING)"
  PRINT #outfile, ""
  PRINT #outfile, "'Here User defined Declarations are added."
  PRINT #outfile, ""
  PRINT #outfile, "DIM SHARED sentence AS STRING"
  PRINT #outfile, "DIM SHARED lastch AS STRING"
  PRINT #outfile, "DIM SHARED ch AS STRING"
  PRINT #outfile, "DIM SHARED nextch AS STRING"
  PRINT #outfile, "DIM SHARED file AS STRING * 8"
  PRINT #outfile, "DIM SHARED path AS STRING"
  PRINT #outfile, ""
  PRINT #outfile, "CONST FALSE = 0"
  PRINT #outfile, "CONST TRUE = NOT FALSE"
  PRINT #outfile, "CONST syntaxfile = 1"
  PRINT #outfile, ""
  PRINT #outfile, "CONST COMMAND = "; CHR$(34); "..."; CHR$(34); "          "; "'"; "Q-BASIC
Version only!"
  PRINT #outfile, ""
  PRINT #outfile, "BEGIN"
  PRINT #outfile, "  WHILE NOT EOF(syntaxfile)"
  PRINT #outfile, "    getsentence"
  PRINT #outfile, "    getword"
  PRINT #outfile, "    IF NOT EOF(syntaxfile) THEN"
  PRINT #outfile, "      WHILE ch = "; CHR$(34); CHR$(34)
  PRINT #outfile, "        getword"
  PRINT #outfile, "      WEND"
  PRINT #outfile, "      "; COMMAND
  PRINT #outfile, "    END IF"
  PRINT #outfile, "  WEND"
  PRINT #outfile, "FIN"
  PRINT #outfile, ""
  PRINT #outfile, "SUB BEGIN"
  PRINT #outfile, "  semantic = 1"
  PRINT #outfile, "  CLS"
  PRINT #outfile, "  file = COMMAND'$"
  PRINT #outfile, "  path = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  OPEN path + file + "; CHR$(34); ".SYN"; CHR$(34); " FOR INPUT AS syntaxfile"
  PRINT #outfile, " 'Here User defined Preparations are added."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Symbol"
  PRINT #outfile, "  Semantic0"
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB ERR0R"
  PRINT #outfile, "  PRINT "; CHR$(34); "      "; CHR$(34); "; CHR$(27); "; CHR$(34); "SYNTAX
ERROR!"; CHR$(34)
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB FIN"
  PRINT #outfile, " 'Here User defined Operations are added."
  PRINT #outfile, "  CLOSE #syntaxfile"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getsentence"
  PRINT #outfile, "  LINE INPUT #syntaxfile, sentence"
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getword"
  PRINT #outfile, "  lastch = ch"
  PRINT #outfile, "  ch = nextch"
  PRINT #outfile, "  IF INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") = 0 THEN"
  PRINT #outfile, "    cha$ = sentence"
  PRINT #outfile, "    sentence = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  ELSE"
  PRINT #outfile, "    cha$ = LEFT$(sentence, INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") -
1)"
  PRINT #outfile, "    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, "; CHR$(34); "
"; CHR$(34); "))"
  PRINT #outfile, "  END IF"
  PRINT #outfile, "  nextch = cha$"
  PRINT #outfile, "  PRINT ch;"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Words"
  PRINT #outfile, " 'Here the names of the rules of the language are parsed."
  PRINT #outfile, "  Semantic1"
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "FUNCTION chINfirst (char AS STRING)"
  PRINT #outfile, "  chINfirst = FALSE"
  PRINT #outfile, "  SELECT CASE char"
  PRINT #outfile, "    CASE "; CHR$(34); "WHILE"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "|"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Words"; CHR$(34); ""
  PRINT #outfile, "      IF NOT (ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34);
"!"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR
ch = "; CHR$(34); "'"; CHR$(34); " OR ch = "; CHR$(34); "[";
  PRINT #outfile, CHR$(34); " OR ch = "; CHR$(34); "]"; CHR$(34); " OR ch = "; CHR$(34); "{";
CHR$(34); " OR ch = "; CHR$(34); "}"; CHR$(34); " OR ch = "; CHR$(34); "."; CHR$(34); ") THEN"
```

```
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Expression"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "Words"; CHR$(34); " OR chINfirst("; CHR$(34);
"Words"; CHR$(34); ") OR ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$
(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = ";
  PRINT #outfile, CHR$(34); "["; CHR$(34); " OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Term"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Factor"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "etc."; CHR$(34); "  'Add user specific conditions"
  PRINT #outfile, "      IF ch = "; CHR$(34); "first('etc.')"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "  END SELECT"
  PRINT #outfile, "END FUNCTION"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Semantic0"
  PRINT #outfile, " 'Here the semantic of "; CHR$(34); "Symbol"; CHR$(34); " is defined."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Semantic1"
  PRINT #outfile, " 'Here the semantic of "; CHR$(34); "Words"; CHR$(34); " is defined."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
END SUB

SUB BNF
  Words
  IF ch = "=" THEN
    Semantic1
    getword
  ELSE
    ERR0R
  END IF
  Expression
  IF ch = "." THEN
    Semantic2
    getword
  ELSE
    ERR0R
  END IF
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB Expression
  IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
      Semantic3
      getword
      Term
    WEND
    IF ch = "!" THEN
      Semantic4
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB Factor
  IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
    Symbol
  ELSEIF ch = "`" THEN
    Semantic5
    getword
    Symbol
    IF ch = "'" THEN
      Semantic6
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "[" THEN
    Semantic7
    getword
    Expression
    IF ch = "]" THEN
      Semantic8
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "{" THEN
    Semantic9
    getword
    Expression
    IF ch = "}" THEN
      Semantic10
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB FIN
  FOR i = 2 TO Semantic
    PRINT #outfile, "DECLARE SUB semantic"; RIGHT$(STR$(i), LEN(STR$(i)) - 1); " ()"
    PRINT #outfile, ""
    PRINT #outfile, "SUB semantic"; RIGHT$(STR$(i), LEN(STR$(i)) - 1)
    PRINT #outfile, " 'Here the semantic of "; CHR$(34); sym(i); CHR$(34); " is defined."
    PRINT #outfile, "END SUB"
    PRINT #outfile, ""
  NEXT i
  CLOSE #outfile
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
  IF lastch = "`" AND nextch = "'" AND ch <> "Symbol" THEN
    IF event = "level" THEN
      PRINT #outfile, "  IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "else" THEN
      PRINT #outfile, "IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "while" THEN
      PRINT #outfile, " ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    END IF
    Semantic = Semantic + 1
    sym(Semantic) = ch
  ELSEIF ch = "Symbol" THEN
    IF event = "else" THEN
      PRINT #outfile, "IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSE
      PRINT #outfile, "    "; ch
    END IF
  END IF
END SUB

SUB Semantic1
```

```
 'Here the semantic of "=" is defined.
  IF ch = "=" THEN
    PRINT #outfile, "DECLARE SUB "; lastch; " ()"
    PRINT #outfile, ""
    PRINT #outfile, "SUB "; lastch
    event = "level"
  END IF
END SUB

SUB Semantic10
 'Here the semantic of "}" is defined.
  IF ch = "}" THEN
    PRINT #outfile, "  WEND"
  END IF
END SUB

SUB Semantic11
 'Here the semantic of "Words" is defined.
  IF event = "level" THEN
    PRINT #outfile, "  "; ch
  ELSEIF event = "while" THEN
    PRINT #outfile, " chINfirst("; CHR$(34); ch; CHR$(34); ") "
    PRINT #outfile, "    "; ch
  ELSEIF event = "else" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); ch; CHR$(34); ") THEN"
    PRINT #outfile, "    "; ch
  END IF
END SUB

SUB Semantic2
 'Here the semantic of "." is defined.
  IF ch = "." THEN
    PRINT #outfile, "END SUB"
    PRINT #outfile, ""
    event = "end"
  END IF
END SUB

SUB Semantic3
 'Here the semantic of "|" is defined.
  IF ch = "|" AND lastch = "=" THEN
    PRINT #outfile, "  ";
    event = "else"
  ELSEIF ch = "|" THEN
    PRINT #outfile, "  ELSE";
    event = "else"
  END IF
END SUB

SUB Semantic4
 'Here the semantic of "!" is defined.
  IF ch = "!" AND nextch = "." THEN
```

```
    PRINT #outfile, "  ELSE"
    PRINT #outfile, "    ERR0R"
    PRINT #outfile, "  END IF"
  END IF
END SUB

SUB Semantic5
 'Here the semantic of "`" is defined.
  IF ch = "`" THEN
  END IF
END SUB

SUB Semantic6
 'Here the semantic of "'" is defined.
  IF ch = "'" THEN
    IF event = "else" THEN
      PRINT #outfile, " THEN"
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "while" THEN
      PRINT #outfile, ""
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "level" THEN
      PRINT #outfile, " THEN"
      PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
      PRINT #outfile, "    getword"
      PRINT #outfile, "  ELSE"
      PRINT #outfile, "    ERR0R"
      PRINT #outfile, "  ENDIF"
    END IF
    event = "level"
  END IF
END SUB

SUB Semantic7
 'Here the semantic of "[" is defined.
  IF ch = "[" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); nextch; CHR$(34); ") THEN"
  END IF
END SUB

SUB Semantic8
 'Here the semantic of "]" is defined.
  IF ch = "]" THEN
    PRINT #outfile, "  END IF"
  END IF
END SUB
```

```
SUB Semantic9
 'Here the semantic of "{" is defined.
  IF lastch = "|" AND ch = "{" AND nextch = "`" THEN
    PRINT #outfile, "IF chINfirst("; CHR$(34); "WHILE"; CHR$(34); ") THEN"
    PRINT #outfile, "    WHILE";
    event = "while"
  ELSEIF ch = "{" THEN
    PRINT #outfile, "  WHILE";
    event = "while"
  END IF
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic11
  getword
END SUB
==============
```

Wendet man dieses Programm wiederum auf die BNF an, so erhält man das Grundgerüst des Programms selbst, wobei lediglich die Deklarationen, Vorbereitungen und Beendigungen und natürlich die Semantik der Symbole selbst eingetragen werden muß, damit es dieselbe Funktionalität zeigt.

Damit hat man den anfangs angestrebten Compiler-Compiler, der zur Untersuchung und Ausführung beliebiger formaler Sprachen benutzt werden kann.

```
==============
[File: BNF.CMP]

'BNF-Compiler  (C) by W.-W.Scheuermann  07-24-1993
'Nicht lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst(char AS STRING)

'Here User defined Declarations are added.

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "..."          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN

SUB BEGIN
  semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
 'Here User defined Preparations are added.
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB FIN
 'Here User defined Operations are added.
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic1
  getword
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
```

```
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
END SUB

SUB Semantic1
 'Here the semantic of "Words" is defined.
END SUB

DECLARE SUB BNF ()

SUB BNF
  Words
  IF ch = "=" THEN
    semantic2
    getword
  ELSE
    ERR0R
  ENDIF
  Expression
  IF ch = "." THEN
    semantic3
    getword
  ELSE
    ERR0R
  ENDIF
END SUB

DECLARE SUB Expression ()

SUB Expression
    IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
    semantic4
    getword
  Term
  WEND
  IF ch = "!" THEN
    semantic5
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB Term ()

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB

DECLARE SUB Factor ()

SUB Factor
    IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
    Symbol
  ELSEIF ch = "`" THEN
    semantic6
    getword
    Symbol
  IF ch = "'" THEN
    semantic7
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "[" THEN
    semantic8
    getword
  Expression
  IF ch = "]" THEN
    semantic9
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "{" THEN
    semantic10
    getword
  Expression
  IF ch = "}" THEN
```

```
    semantic11
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB semantic2 ()

SUB semantic2
 'Here the semantic of "=" is defined.
END SUB

DECLARE SUB semantic3 ()

SUB semantic3
 'Here the semantic of "." is defined.
END SUB

DECLARE SUB semantic4 ()

SUB semantic4
 'Here the semantic of "|" is defined.
END SUB

DECLARE SUB semantic5 ()

SUB semantic5
 'Here the semantic of "!" is defined.
END SUB

DECLARE SUB semantic6 ()

SUB semantic6
 'Here the semantic of "`" is defined.
END SUB

DECLARE SUB semantic7 ()

SUB semantic7
 'Here the semantic of "'" is defined.
END SUB

DECLARE SUB semantic8 ()

SUB semantic8
 'Here the semantic of "[" is defined.
END SUB
DECLARE SUB semantic9 ()

SUB semantic9
 'Here the semantic of "]" is defined.
END SUB

DECLARE SUB semantic10 ()

SUB semantic10
 'Here the semantic of "{" is defined.
END SUB

DECLARE SUB semantic11 ()

SUB semantic11
 'Here the semantic of "}" is defined.
END SUB
==============
```

Um die Korrektheit des Programms zu beweisen gehen wir nun folgendermaßen vor:

Wir übertragen ganz formal die semantischen Anweisungen aus BNF1.BAS nach BNF.CMP und erhalten den fertigen BNF-Compiler BNF1.CMP indem wir die Deklarationen bis zur Anweisung BEGIN kopieren, ebenso die Funktion chINfirst(), sowie alle semantischen Anweisungen in die entsprechenden Subroutinen kopieren. Als einzige Veränderung soll das Output-File BNF.CPL heißen.

```
==============
[File: BNF1.CMP]

'BNF-Compiler  (C) by W.-W.Scheuermann  07-24-1993
'Lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst (char AS STRING)

'Here User defined Declarations are added.

DECLARE SUB BNF ()
DECLARE SUB Expression ()
DECLARE SUB Term ()
DECLARE SUB Factor ()
DECLARE SUB Semantic2 ()
DECLARE SUB Semantic3 ()
DECLARE SUB Semantic4 ()
DECLARE SUB Semantic5 ()
DECLARE SUB Semantic6 ()
DECLARE SUB Semantic7 ()
DECLARE SUB Semantic8 ()
DECLARE SUB Semantic9 ()
DECLARE SUB Semantic10 ()
DECLARE SUB Semantic11 ()

DIM SHARED event AS STRING
DIM SHARED sym(1 TO 30) AS STRING
DIM SHARED Semantic AS INTEGER

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST outfile = 2

CONST COMMAND = "BNF"          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN

SUB BEGIN
  Semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
 'Here User defined Preparations are added.
  OPEN path + file + ".CPL" FOR OUTPUT AS outfile
  event = "end"
  PRINT #outfile, "'"; COMMAND; "-Compiler  (C) by W.-W.Scheuermann  ";
  PRINT #outfile, DATE$
  PRINT #outfile, "'Nicht lauffähig!"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE SUB Semantic0 ()"
  PRINT #outfile, "DECLARE SUB Semantic1 ()"
  PRINT #outfile, "DECLARE SUB Symbol ()"
  PRINT #outfile, "DECLARE SUB Words ()"
  PRINT #outfile, "DECLARE SUB BEGIN ()"
  PRINT #outfile, "DECLARE SUB ERR0R ()"
  PRINT #outfile, "DECLARE SUB FIN ()"
  PRINT #outfile, "DECLARE SUB getsentence ()"
  PRINT #outfile, "DECLARE SUB getword ()"
  PRINT #outfile, ""
  PRINT #outfile, "DECLARE FUNCTION chINfirst(char AS STRING)"
  PRINT #outfile, ""
  PRINT #outfile, "'Here User defined Declarations are added."
  PRINT #outfile, ""
  PRINT #outfile, "DIM SHARED sentence AS STRING"
  PRINT #outfile, "DIM SHARED lastch AS STRING"
  PRINT #outfile, "DIM SHARED ch AS STRING"
  PRINT #outfile, "DIM SHARED nextch AS STRING"
  PRINT #outfile, "DIM SHARED file AS STRING * 8"
  PRINT #outfile, "DIM SHARED path AS STRING"
  PRINT #outfile, ""
  PRINT #outfile, "CONST FALSE = 0"
  PRINT #outfile, "CONST TRUE = NOT FALSE"
  PRINT #outfile, "CONST syntaxfile = 1"
  PRINT #outfile, ""
  PRINT #outfile, "CONST COMMAND = "; CHR$(34); "..."; CHR$(34); "          "; "'"; "Q-BASIC
Version only!"
  PRINT #outfile, ""
  PRINT #outfile, "BEGIN"
  PRINT #outfile, "  WHILE NOT EOF(syntaxfile)"
  PRINT #outfile, "    getsentence"
  PRINT #outfile, "    getword"
  PRINT #outfile, "    IF NOT EOF(syntaxfile) THEN"
  PRINT #outfile, "      WHILE ch = "; CHR$(34); CHR$(34)
  PRINT #outfile, "        getword"
  PRINT #outfile, "      WEND"
  PRINT #outfile, "      "; COMMAND
  PRINT #outfile, "    END IF"
  PRINT #outfile, "  WEND"
  PRINT #outfile, "FIN"
  PRINT #outfile, ""
  PRINT #outfile, "SUB BEGIN"
  PRINT #outfile, "  semantic = 1"
  PRINT #outfile, "  CLS"
  PRINT #outfile, "  file = COMMAND'$"
  PRINT #outfile, "  path = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  OPEN path + file + "; CHR$(34); ".SYN"; CHR$(34); " FOR INPUT AS syntaxfile"
  PRINT #outfile, " 'Here User defined Preparations are added."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Symbol"
  PRINT #outfile, "  Semantic0"
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB ERR0R"
  PRINT #outfile, "  PRINT "; CHR$(34); "      "; CHR$(34); "; CHR$(27); "; CHR$(34); "SYNTAX
ERROR!"; CHR$(34)
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB FIN"
  PRINT #outfile, " 'Here User defined Operations are added."
  PRINT #outfile, "  CLOSE #syntaxfile"
  PRINT #outfile, "  END"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getsentence"
  PRINT #outfile, "  LINE INPUT #syntaxfile, sentence"
  PRINT #outfile, "  PRINT"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB getword"
  PRINT #outfile, "  lastch = ch"
  PRINT #outfile, "  ch = nextch"
  PRINT #outfile, "  IF INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") = 0 THEN"
  PRINT #outfile, "    cha$ = sentence"
  PRINT #outfile, "    sentence = "; CHR$(34); CHR$(34)
  PRINT #outfile, "  ELSE"
  PRINT #outfile, "    cha$ = LEFT$(sentence, INSTR(sentence, "; CHR$(34); " "; CHR$(34); ") -
1)"
  PRINT #outfile, "    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, "; CHR$(34); "
"; CHR$(34); "))"
  PRINT #outfile, "  END IF"
  PRINT #outfile, "  nextch = cha$"
  PRINT #outfile, "  PRINT ch;"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Words"
  PRINT #outfile, " 'Here the names of the rules of the language are parsed."
  PRINT #outfile, "  Semantic1"
  PRINT #outfile, "  getword"
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "FUNCTION chINfirst (char AS STRING)"
  PRINT #outfile, "  chINfirst = FALSE"
  PRINT #outfile, "  SELECT CASE char"
  PRINT #outfile, "    CASE "; CHR$(34); "WHILE"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "|"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Words"; CHR$(34); ""
  PRINT #outfile, "      IF NOT (ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34);
"!"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR
ch = "; CHR$(34); "'"; CHR$(34); " OR ch = "; CHR$(34); "[";
  PRINT #outfile, CHR$(34); " OR ch = "; CHR$(34); "]"; CHR$(34); " OR ch = "; CHR$(34); "{";
CHR$(34); " OR ch = "; CHR$(34); "}"; CHR$(34); " OR ch = "; CHR$(34); "."; CHR$(34); ") THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Expression"; CHR$(34); ""
  PRINT #outfile, "      IF ch = "; CHR$(34); "Words"; CHR$(34); " OR chINfirst("; CHR$(34);
"Words"; CHR$(34); ") OR ch = "; CHR$(34); "Symbol"; CHR$(34); " OR ch = "; CHR$(34); "|"; CHR$
(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = ";
  PRINT #outfile, CHR$(34); "["; CHR$(34); " OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Term"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "Factor"; CHR$(34); ""
  PRINT #outfile, "      IF chINfirst("; CHR$(34); "Words"; CHR$(34); ") OR ch = "; CHR$(34);
"Symbol"; CHR$(34); " OR ch = "; CHR$(34); "`"; CHR$(34); " OR ch = "; CHR$(34); "["; CHR$(34); "
OR ch = "; CHR$(34); "{"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "    CASE "; CHR$(34); "etc."; CHR$(34); "  'Add user specific conditions"
  PRINT #outfile, "      IF ch = "; CHR$(34); "first('etc.')"; CHR$(34); " THEN"
  PRINT #outfile, "        chINfirst = TRUE"
  PRINT #outfile, "      END IF"
  PRINT #outfile, "  END SELECT"
  PRINT #outfile, "END FUNCTION"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Semantic0"
  PRINT #outfile, " 'Here the semantic of "; CHR$(34); "Symbol"; CHR$(34); " is defined."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
  PRINT #outfile, "SUB Semantic1"
  PRINT #outfile, " 'Here the semantic of "; CHR$(34); "Words"; CHR$(34); " is defined."
  PRINT #outfile, "END SUB"
  PRINT #outfile, ""
END SUB

SUB BNF
  Words
  IF ch = "=" THEN
    Semantic2
    getword
  ELSE
```

```
    ERR0R
  END IF
  Expression
  IF ch = "." THEN
    Semantic3
    getword
  ELSE
    ERR0R
  END IF
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB Expression
  IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
      Semantic4
      getword
      Term
    WEND
    IF ch = "!" THEN
      Semantic5
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB Factor
  IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
    Symbol
  ELSEIF ch = "`" THEN
    Semantic6
    getword
    Symbol
    IF ch = "'" THEN
      Semantic7
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "[" THEN
    Semantic8
    getword
    Expression
    IF ch = "]" THEN
      Semantic9
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF ch = "{" THEN
    Semantic10
    getword
    Expression
    IF ch = "}" THEN
      Semantic11
      getword
    ELSE
      ERR0R
    END IF
  ELSE
    ERR0R
  END IF
END SUB

SUB FIN
 'Here User defined Operations are added.
  FOR i = 2 TO Semantic
    PRINT #outfile, "DECLARE SUB semantic"; RIGHT$(STR$(i), LEN(STR$(i)) - 1); " ()"
    PRINT #outfile, ""
    PRINT #outfile, "SUB semantic"; RIGHT$(STR$(i), LEN(STR$(i)) - 1)
    PRINT #outfile, " 'Here the semantic of "; CHR$(34); sym(i); CHR$(34); " is defined."
    PRINT #outfile, "END SUB"
    PRINT #outfile, ""
  NEXT i
  CLOSE #outfile
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
  IF lastch = "`" AND nextch = "'" AND ch <> "Symbol" THEN
    IF event = "level" THEN
      PRINT #outfile, "  IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "else" THEN
      PRINT #outfile, "IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSEIF event = "while" THEN
      PRINT #outfile, " ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    END IF
    Semantic = Semantic + 1
    sym(Semantic) = ch
  ELSEIF ch = "Symbol" THEN
```

```
    IF event = "else" THEN
      PRINT #outfile, "IF ch = "; CHR$(34); ""; ch; ""; CHR$(34); "";
    ELSE
      PRINT #outfile, "    "; ch
    END IF
  END IF
END SUB

SUB Semantic1
 'Here the semantic of "Words" is defined.
  IF event = "level" THEN
    PRINT #outfile, "  "; ch
  ELSEIF event = "while" THEN
    PRINT #outfile, " chINfirst("; CHR$(34); ch; CHR$(34); ") "
    PRINT #outfile, "    "; ch
  ELSEIF event = "else" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); ch; CHR$(34); ") THEN"
    PRINT #outfile, "    "; ch
  END IF
END SUB

SUB Semantic10
 'Here the semantic of "{" is defined.
  IF lastch = "|" AND ch = "{" AND nextch = "`" THEN
    PRINT #outfile, "IF chINfirst("; CHR$(34); "WHILE"; CHR$(34); ") THEN"
    PRINT #outfile, "    WHILE";
    event = "while"
  ELSEIF ch = "{" THEN
    PRINT #outfile, "  WHILE";
    event = "while"
  END IF
END SUB

SUB Semantic11
 'Here the semantic of "}" is defined.
  IF ch = "}" THEN
    PRINT #outfile, "  WEND"
  END IF
END SUB

SUB Semantic2
 'Here the semantic of "=" is defined.
  IF ch = "=" THEN
    PRINT #outfile, "DECLARE SUB "; lastch; " ()"
    PRINT #outfile, ""
    PRINT #outfile, "SUB "; lastch
    event = "level"
  END IF
END SUB

SUB Semantic3
 'Here the semantic of "." is defined.
```

```
  IF ch = "." THEN
    PRINT #outfile, "END SUB"
    PRINT #outfile, ""
    event = "end"
  END IF
END SUB

SUB Semantic4
 'Here the semantic of "|" is defined.
  IF ch = "|" AND lastch = "=" THEN
    PRINT #outfile, "  ";
    event = "else"
  ELSEIF ch = "|" THEN
    PRINT #outfile, "  ELSE";
    event = "else"
  END IF
END SUB

SUB Semantic5
 'Here the semantic of "!" is defined.
  IF ch = "!" AND nextch = "." THEN
    PRINT #outfile, "  ELSE"
    PRINT #outfile, "    ERR0R"
    PRINT #outfile, "  END IF"
  END IF
END SUB

SUB Semantic6
 'Here the semantic of "`" is defined.
  IF ch = "`" THEN
  END IF
END SUB

SUB Semantic7
 'Here the semantic of "'" is defined.
  IF ch = "'" THEN
    IF event = "else" THEN
      PRINT #outfile, " THEN"
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "while" THEN
      PRINT #outfile, ""
      IF lastch <> "Symbol" THEN
        PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
        PRINT #outfile, "    getword"
      END IF
    ELSEIF event = "level" THEN
      PRINT #outfile, " THEN"
      PRINT #outfile, "    semantic"; RIGHT$(STR$(Semantic), LEN(STR$(Semantic)) - 1)
      PRINT #outfile, "    getword"
      PRINT #outfile, "  ELSE"
      PRINT #outfile, "    ERR0R"
      PRINT #outfile, "  ENDIF"
    END IF
    event = "level"
  END IF
END SUB

SUB Semantic8
 'Here the semantic of "[" is defined.
  IF ch = "[" THEN
    PRINT #outfile, "  IF chINfirst("; CHR$(34); nextch; CHR$(34); ") THEN"
  END IF
END SUB

SUB Semantic9
 'Here the semantic of "]" is defined.
  IF ch = "]" THEN
    PRINT #outfile, "  END IF"
  END IF
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic1
  getword
END SUB
==============
[File: BNF.CPL]

'BNF-Compiler  (C) by W.-W.Scheuermann  08-02-1993
'Nicht lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst(char AS STRING)

'Here User defined Declarations are added.

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "..."          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      BNF
    END IF
  WEND
FIN

SUB BEGIN
  semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
 'Here User defined Preparations are added.
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB
```

```
SUB FIN
 'Here User defined Operations are added.
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic1
  getword
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
END SUB

SUB Semantic1
 'Here the semantic of "Words" is defined.
END SUB

DECLARE SUB BNF ()

SUB BNF
  Words
  IF ch = "=" THEN
    semantic2
    getword
  ELSE
    ERR0R
  ENDIF
  Expression
  IF ch = "." THEN
    semantic3
    getword
  ELSE
    ERR0R
  ENDIF
END SUB

DECLARE SUB Expression ()

SUB Expression
    IF chINfirst("Term") THEN
    Term
  ELSEIF chINfirst("WHILE") THEN
    WHILE ch = "|"
    semantic4
    getword
  Term
  WEND
  IF ch = "!" THEN
    semantic5
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB Term ()

SUB Term
  Factor
  WHILE chINfirst("Factor")
    Factor
  WEND
END SUB

DECLARE SUB Factor ()

SUB Factor
    IF chINfirst("Words") THEN
    Words
  ELSEIF ch = "Symbol" THEN
    Symbol
  ELSEIF ch = "`" THEN
    semantic6
    getword
    Symbol
  IF ch = "'" THEN
    semantic7
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "[" THEN
    semantic8
    getword
  Expression
  IF ch = "]" THEN
    semantic9
    getword
  ELSE
    ERR0R
  ENDIF
  ELSEIF ch = "{" THEN
    semantic10
    getword
  Expression
  IF ch = "}" THEN
    semantic11
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB semantic2 ()

SUB semantic2
 'Here the semantic of "=" is defined.
END SUB

DECLARE SUB semantic3 ()

SUB semantic3
 'Here the semantic of "." is defined.
END SUB

DECLARE SUB semantic4 ()

SUB semantic4
 'Here the semantic of "|" is defined.
END SUB

DECLARE SUB semantic5 ()

SUB semantic5
 'Here the semantic of "!" is defined.
END SUB

DECLARE SUB semantic6 ()

SUB semantic6
 'Here the semantic of "`" is defined.
END SUB

DECLARE SUB semantic7 ()

SUB semantic7
 'Here the semantic of "'" is defined.
END SUB

DECLARE SUB semantic8 ()

SUB semantic8
 'Here the semantic of "[" is defined.
END SUB

DECLARE SUB semantic9 ()
SUB semantic9
 'Here the semantic of "]" is defined.
END SUB

DECLARE SUB semantic10 ()

SUB semantic10
 'Here the semantic of "{" is defined.
END SUB

DECLARE SUB semantic11 ()

SUB semantic11
 'Here the semantic of "}" is defined.
END SUB
==============
```

Wir vergleichen nun den Output der beiden Compiler BNF1.BAS und BNF1.CMP, nämlich die Files BNF.CMP und BNF.CPL. Da BNF.CMP die Basisstruktur für BNF1.CMP ist, der wiederum BNF.CPL erzeugt, beweist die Gleichheit von BNF.CMP und BNF.CPL die Korrektheit des Programms BNF1.CMP. Das Programm hat sich selbst reproduziert. <Neunter Selbstbezug>

Dies ist übrigens der einzige streng formale Programmbeweis, den ich kenne. Allerdings gilt der formale Beweis, und das ist leider eine gravierende Einschränkung, nur für die Teile des Programms, die von einem anderen Programm erzeugt worden sind und nicht für unsere von Hand durchgeführten Übertragungen. Aber immerhin ist es besser als nichts: BNF1.CMP ist unser Compiler-Compiler!

```
==============
[File: VERGLICH.TXT]

>comp BNF.CMP BNF.CPL
Vergleiche BNF.CMP und BNF.CPL...
Dateien identisch

Weitere Dateien vergleichen (J/N)? n

>
==============
```

Wenn der Compiler einen Syntaxfehler gemäß der Backus-Naur-Form erkennt, meldet er dies. Als Beispiel sei ein Fehler in die BNF-Syntax eingebaut. Der Compiler BNF1.CMP erzeugt folgende Fehlermeldung bei der Anzeige der übersetzten Syntax:

```
==============
[BNF.ERR]

BNF=Words`='Expression`.'.
Expression=|Term|{`|'Term}`!'!.
Term=Factor{Factor).      ←SYNTAX ERROR!
==============
```

Nun gilt es noch zu zeigen, daß unser Compiler auch für andere Sprachen geeignet ist.

Als Beispiel wähle ich Lindenmayer-Systeme. Dies ist eine Sprache zur Beschreibung pflanzlischer Formen. Die Sätze dieser Sprache beschreiben die grafische Darstellung genau einer Wachstumsform, wobei die terminalen Symbole als Plottanweisungen für Turtle-Graphiken interpretiert werden. Die Sätze können in einer komprimierten Form beschrieben werden:

```
==============
n=4,d=22.5°
X
X->F-[[X]+X]+F[+FX]-X
F->FF
==============
```

Hier soll jedoch die Langform verwendet werden:

```
==============
[LSATZ.TEX:Teil]

d=22.5,FF-[-F+F+F]+[+F-F-F]FF-[-F+F+F]+[+F-F-F]-[-FF-[-F+F+F]+[+F-F-F]+FF-[-F
+F+F]+[+F-F-F]+FF-[-F+F]+[+F-F-F]FF-[-F+F+F]+[+F-F-F]-[-FF-[-F+F+F]+[+F-F-F]
+FF-[-F+F+F]+[+F-F-F]+FF-[-F+F+F]+[+F-F-F]]+[+FF-[-F+F+F]+[+F-F-F]-FF-[-F+F+F]
+[+F-F-F]-FF-[-F+F+F]+[+F-F-F]];
==============
```

Dieser Satz muß für den Lindenmayer-Compiler zum einfachen Scannen mit Leerzeichen und Klammeraffe aufbereitet werden:

```
==============
[LSYSTEM.TXT]

d= 2 2 . 5 , F F F F F F F F F F F F F F F F [ [ F F F F F F F F [ [ F F F F [ [ F F [ [ X ] +
X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F
[ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ [ F
F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ +
F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F
F F F F F F F [ + F F F F F F F F F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [
X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ]
- F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X
] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F
F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F F F F F [ [ F F F F [ [ F F [ [
X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F
F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F
[ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F
F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X
] + F F F F F F F F [ + F F F F F F F F F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F
F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ]
- X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X
] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F
[ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F F F F F F F F F F F F F [
+ F F F F F F F F F F F F F F F F F F F F F F F F [ [ F F F F [ [ F F [ [ X ] + X ] + F F [ + F F
X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F
F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ [ F F [ [ X ] + X ] +
F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X
] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F F F F F [ +
F F F F F F F F F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [
+ F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X
] + F F [ + F F X ] - X ] - F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] +
X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F
[ [ X ] + X ] + F F [ + F F X ] - X ] - F F F F F F F F [ [ F F F F [ [ F F [ [ X ] + X ] + F F [
+ F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X
] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ [ F F [ [ X ] +
X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F
[ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F F F F F F
F [ + F F F F F F F F F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [ X ] + X ] +
F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ] - F F [ [ X
] + X ] + F F [ + F F X ] - X ] - F F F F [ [ F F [ [ X ] + X ] + F F [ + F F X ] - X ] + F F [ [
X ] + X ] + F F [ + F F X ] - X ] + F F F F [ + F F F F F F [ [ X ] + X ] + F F [ + F F X ] - X ]
- F F [ [ X ] + X ] + F F [ + F F X ] - X ; @
==============
```

Um unkompliziert die Langform der Sätze aus der komprimierten erzeugen zu können, habe ich ein kurzes Programm geschrieben:

```
==============
[LSATZ.BAS]
```

```
DIM saz AS STRING
DIM saz1 AS STRING
CLS
INPUT "Axiom?       ", s0$
INPUT "First term?  ", s01$
INPUT "                 ->  ", s1$
INPUT "Second term? ", s02$
INPUT "                 ->  ", s2$
INPUT "Angle?       ", d$
INPUT "Iteration?   ", jj
OPEN "lsystem.txt" FOR OUTPUT AS #1
  saz = s0$
  FOR j = 1 TO jj
    saz1 = ""
    FOR i = 1 TO 3000
      IF (MID$(saz, i, 1) = s01$) THEN
        saz1 = saz1 + s1$
      ELSE
        saz1 = saz1 + MID$(saz, i, 1)
      END IF
    NEXT i
    saz = saz1
    saz1 = ""
    FOR i = 1 TO 3000
      IF (MID$(saz, i, 1) = s02$) THEN
        saz1 = saz1 + s2$
      ELSE
        saz1 = saz1 + MID$(saz, i, 1)
      END IF
    NEXT i
    saz = saz1
    PRINT
    PRINT j; " "; saz
  NEXT j
  PRINT #1, "d= " + LEFT$(d$, 1) + " " + MID$(d$, 2, 1) + " . " + RIGHT$(d$, 1) + " , ";
  FOR i = 1 TO LEN(saz1)
    PRINT #1, MID$(saz1, i, 1) + " ";
  NEXT i
  PRINT #1, "@ "
  PRINT #1, ""
CLOSE #1
==================
```

Dieses Programm erzeugt aus der Kurzform die Langform eines Satzes:

```
==================
[LSATZ.TEX]

Axiom?       X
First term?  X
                ->  F[[X]+X]+F[+FX]-X
```

```
Second term? F
                ->  FF
Angle?       22.5
Iteration?   4

 1  FF[[X]+X]+FF[+FFX]-X

 2
FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF
[+FFX]-X

 3
FFFFFFFF[[FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF
[[X]+X]+FF[+FFX]-X]+FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF
[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[+FFFFFFFFFFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-
X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]-FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF
[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X

 4
FFFFFFFFFFFFFFFF[[FFFFFFFF[[FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]
+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF
[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[+FFFFFFFFFFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]
+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]-FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF
[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[[FFFF[[FF[[X]
+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+
FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]
+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[+FFFFFFFFFFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF
[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]-FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF
[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFFFFFFFFFF
[+FFFFFFFFFFFFFFFFFFFFFFFF[[FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]
+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF
[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[+FFFFFFFFFFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]
+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF
[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]-FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF
[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]-FFFFFFFF[[FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF
[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF
[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-X]+FFFFFFFF[+FFFFFFFFFFFF
[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF[+FFX]-
X]-FFFF[[FF[[X]+X]+FF[+FFX]-X]+FF[[X]+X]+FF[+FFX]-X]+FFFF[+FFFFFF[[X]+X]+FF[+FFX]-X]-FF[[X]+X]+FF
[+FFX]-X
==============
```

Die Semantik der terminalen Symbole dieser Sprache ist folgende:

"

F Move forward a step of length L. The state of the turtle changes to (x',y',alpha) where x'=x+L*cos alpha and y'=y+L*sin alpha. A line segment between points (x,y) and (x',y') is drawn.

f Move foreward a step of length L without drawing a line.

+ Turn left by angle d. The next state of the turtle is (x,y,alpha+d). The positive orientation of angles is counterclockwise.

- Turn right by angle d. The next state of the turtle is (x,y,alpha-d).

l,r,Xare ignored.

[ Push the current state of the turtle onto a pushdown stack. The information saved on the stack contains the turtle's position and orientation, and possibly other attributes such as the colour and width of lines being drawn.

] Pop a state from the stack and make it the current state of the turtle. No line is drawn, although in general the position of the turtle changes.
"
Diese Semantik muß also in den automatisch zu erzeugenden Parser implementiert werden, um einen funktionsfähigen Compiler zu erhalten. Zuerst muß wieder die Syntax der Sprache definiert werden:

```
==============
[LSYSTEMS.TEX]

LSYSTEM=Deklaration`,'Produktion`;'.
Deklaration=`d='ZifferZiffer`.'Ziffer.
Produktion=`F'{Ausdruck}.
Befehl=Ausdruck{Ausdruck}.
Ausdruck=|`+'Ausdruck|`-'Ausdruck|`['Befehl`]'|Anweisung!.
Ziffer=|`0'|`1'|`2'|`3'|`4'|`5'|`6'|`7'|`8'|`9'!.
Anweisung=|`F'|`Fr'|`Fl'|`f'|`X'|`r'|`l'!.@
==============
```

Daraus wird wie gehabt die maschinenlesbare Form:

```
==============
[LSYSTEM.SYN]

LSYSTEM = Deklaration ` , ' Produktion ` ; ' .
Deklaration = ` d= ' Ziffer Ziffer ` . ' Ziffer .
Produktion = ` F ' { Ausdruck } .
Befehl = Ausdruck { Ausdruck } .
Ausdruck = | ` + ' Ausdruck | ` - ' Ausdruck | ` [ ' Befehl ` ] ' | Anweisung ! .
Ziffer = | ` 0 ' | ` 1 ' | ` 2 ' | ` 3 ' | ` 4 ' | ` 5 ' | ` 6 ' | ` 7 ' | ` 8 ' | ` 9 ' ! .
Anweisung = | ` F ' | ` Fr ' | ` Fl ' | ` f ' | ` X ' | ` r ' | ` l ' ! . @
==============
```

Indem wir unseren Compiler-Compiler BNF1.CMP laufen lassen erhalten wir den Lindenmayer Parser, der, nach Ergänzung der Funktion `chINfirst()`, immerhin bereits in der Lage ist zu prüfen, ob unsere Lindenmayer-Sätze syntaktisch korrekt sind:

```
==============
[LSYSTEM.CPL]

'LSYSTEM-Compiler  (C) by W.-W.Scheuermann  08-05-1993
'Nicht lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst(char AS STRING)

'Here User defined Declarations are added.

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "..."          'Q-BASIC Version only!

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      LSYSTEM
    END IF
  WEND
FIN

SUB BEGIN
  semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".SYN" FOR INPUT AS syntaxfile
```

```
 'Here User defined Preparations are added.
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB FIN
 'Here User defined Operations are added.
  CLOSE #syntaxfile
  END
END SUB

SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
  PRINT ch;
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic1
  getword
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
END SUB

SUB Semantic1
 'Here the semantic of "Words" is defined.
END SUB

DECLARE SUB LSYSTEM ()

SUB LSYSTEM
  Deklaration
  IF ch = "," THEN
    semantic2
    getword
  ELSE
    ERR0R
  ENDIF
  Produktion
  IF ch = ";" THEN
    semantic3
    getword
  ELSE
    ERR0R
  ENDIF
END SUB
```

```
DECLARE SUB Deklaration ()

SUB Deklaration
  IF ch = "d=" THEN
    semantic4
    getword
  ELSE
    ERR0R
  ENDIF
  Ziffer
  Ziffer
  IF ch = "." THEN
    semantic5
    getword
  ELSE
    ERR0R
  ENDIF
  Ziffer
END SUB

DECLARE SUB Produktion ()

SUB Produktion
  IF ch = "F" THEN
    semantic6
    getword
  ELSE
    ERR0R
  ENDIF
  WHILE chINfirst("Ausdruck")
    Ausdruck
  WEND
END SUB

DECLARE SUB Befehl ()

SUB Befehl
  Ausdruck
  WHILE chINfirst("Ausdruck")
    Ausdruck
  WEND
END SUB

DECLARE SUB Ausdruck ()

SUB Ausdruck
  IF ch = "+" THEN
    semantic7
    getword
  Ausdruck
  ELSEIF ch = "-" THEN
    semantic8
    getword
  Ausdruck
  ELSEIF ch = "[" THEN
    semantic9
    getword
  Befehl
  IF ch = "]" THEN
    semantic10
    getword
  ELSE
    ERR0R
  ENDIF
  ELSE  IF chINfirst("Anweisung") THEN
    Anweisung
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB Ziffer ()

SUB Ziffer
  IF ch = "0" THEN
    semantic11
    getword
  ELSEIF ch = "1" THEN
    semantic12
    getword
  ELSEIF ch = "2" THEN
    semantic13
    getword
  ELSEIF ch = "3" THEN
    semantic14
    getword
  ELSEIF ch = "4" THEN
    semantic15
    getword
  ELSEIF ch = "5" THEN
    semantic16
    getword
  ELSEIF ch = "6" THEN
    semantic17
    getword
  ELSEIF ch = "7" THEN
    semantic18
    getword
  ELSEIF ch = "8" THEN
    semantic19
    getword
  ELSEIF ch = "9" THEN
    semantic20
    getword
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB Anweisung ()

SUB Anweisung
  IF ch = "F" THEN
    semantic21
    getword
  ELSEIF ch = "Fr" THEN
    semantic22
    getword
  ELSEIF ch = "Fl" THEN
    semantic23
    getword
  ELSEIF ch = "f" THEN
    semantic24
    getword
  ELSEIF ch = "X" THEN
    semantic25
    getword
  ELSEIF ch = "r" THEN
    semantic26
    getword
  ELSEIF ch = "l" THEN
    semantic27
    getword
  ELSE
    ERR0R
  END IF
END SUB

DECLARE SUB semantic2 ()

SUB semantic2
 'Here the semantic of "," is defined.
END SUB

DECLARE SUB semantic3 ()

SUB semantic3
 'Here the semantic of ";" is defined.
END SUB

DECLARE SUB semantic4 ()

SUB semantic4
 'Here the semantic of "d=" is defined.
END SUB
```

```
DECLARE SUB semantic5 ()

SUB semantic5
 'Here the semantic of "." is defined.
END SUB

DECLARE SUB semantic6 ()

SUB semantic6
 'Here the semantic of "F" is defined.
END SUB

DECLARE SUB semantic7 ()

SUB semantic7
 'Here the semantic of "+" is defined.
END SUB

DECLARE SUB semantic8 ()

SUB semantic8
 'Here the semantic of "-" is defined.
END SUB

DECLARE SUB semantic9 ()

SUB semantic9
 'Here the semantic of "[" is defined.
END SUB

DECLARE SUB semantic10 ()

SUB semantic10
 'Here the semantic of "]" is defined.
END SUB

DECLARE SUB semantic11 ()

SUB semantic11
 'Here the semantic of "0" is defined.
END SUB

DECLARE SUB semantic12 ()

SUB semantic12
 'Here the semantic of "1" is defined.
END SUB

DECLARE SUB semantic13 ()

SUB semantic13
 'Here the semantic of "2" is defined.
```

```
END SUB

DECLARE SUB semantic14 ()

SUB semantic14
 'Here the semantic of "3" is defined.
END SUB

DECLARE SUB semantic15 ()

SUB semantic15
 'Here the semantic of "4" is defined.
END SUB

DECLARE SUB semantic16 ()

SUB semantic16
 'Here the semantic of "5" is defined.
END SUB

DECLARE SUB semantic17 ()

SUB semantic17
 'Here the semantic of "6" is defined.
END SUB

DECLARE SUB semantic18 ()

SUB semantic18
 'Here the semantic of "7" is defined.
END SUB

DECLARE SUB semantic19 ()

SUB semantic19
 'Here the semantic of "8" is defined.
END SUB

DECLARE SUB semantic20 ()

SUB semantic20
 'Here the semantic of "9" is defined.
END SUB

DECLARE SUB semantic21 ()

SUB semantic21
 'Here the semantic of "F" is defined.
END SUB

DECLARE SUB semantic22 ()
```

```
SUB semantic22
 'Here the semantic of "Fr" is defined.
END SUB

DECLARE SUB semantic23 ()

SUB semantic23
 'Here the semantic of "Fl" is defined.
END SUB

DECLARE SUB semantic24 ()

SUB semantic24
 'Here the semantic of "f" is defined.
END SUB

DECLARE SUB semantic25 ()

SUB semantic25
 'Here the semantic of "X" is defined.
END SUB

DECLARE SUB semantic26 ()

SUB semantic26
 'Here the semantic of "r" is defined.
END SUB

DECLARE SUB semantic27 ()

SUB semantic27
 'Here the semantic of "l" is defined.
END SUB
==============
```

Nach Umsetzung der Semantik der terminalen Symbole in den dafür vorgesehenen Subroutinen entsprechend der oben angeführten Definitionen erhalten wir den lauffähigen Lindenmayer-Compiler:

```
==============
[LSYSTEM1.CPL]

'LSYSTEM-Compiler  (C) by W.-W.Scheuermann  08-02-1993
'Lauffähig!

DECLARE SUB Semantic0 ()
DECLARE SUB Semantic1 ()
DECLARE SUB Symbol ()
DECLARE SUB Words ()
DECLARE SUB BEGIN ()
DECLARE SUB ERR0R ()
DECLARE SUB FIN ()
DECLARE SUB getsentence ()
DECLARE SUB getword ()

DECLARE FUNCTION chINfirst (char AS STRING)

'Here User defined Declarations are added.

DECLARE SUB LSYSTEM ()
DECLARE SUB Deklaration ()
DECLARE SUB Produktion ()
DECLARE SUB Befehl ()
DECLARE SUB Ausdruck ()
DECLARE SUB Ziffer ()
DECLARE SUB Anweisung ()
DECLARE SUB semantic2 ()
DECLARE SUB semantic3 ()
DECLARE SUB semantic4 ()
DECLARE SUB semantic5 ()
DECLARE SUB semantic6 ()
DECLARE SUB semantic7 ()
DECLARE SUB semantic8 ()
DECLARE SUB semantic9 ()
DECLARE SUB semantic10 ()
DECLARE SUB semantic11 ()
DECLARE SUB semantic12 ()
DECLARE SUB semantic13 ()
DECLARE SUB semantic14 ()
DECLARE SUB semantic15 ()
DECLARE SUB semantic16 ()
DECLARE SUB semantic17 ()
DECLARE SUB semantic18 ()
DECLARE SUB semantic19 ()
DECLARE SUB semantic20 ()
DECLARE SUB semantic21 ()
DECLARE SUB semantic22 ()
DECLARE SUB semantic23 ()
DECLARE SUB semantic24 ()
DECLARE SUB semantic25 ()
DECLARE SUB semantic26 ()
DECLARE SUB semantic27 ()

TYPE Stat
  x AS SINGLE
  y AS SINGLE
  d AS SINGLE
END TYPE

DIM SHARED OldStatus AS Stat
DIM SHARED Status(0 TO 20) AS Stat
DIM SHARED Stack AS INTEGER
DIM SHARED Delta AS SINGLE
DIM SHARED Length AS SINGLE
DIM SHARED Number AS STRING

DIM SHARED sentence AS STRING
DIM SHARED lastch AS STRING
DIM SHARED ch AS STRING
DIM SHARED nextch AS STRING
DIM SHARED file AS STRING * 8
DIM SHARED path AS STRING

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST syntaxfile = 1

CONST COMMAND = "LSYSTEM"          'Q-BASIC Version only!

CONST RAD = .017453292#

BEGIN
  WHILE NOT EOF(syntaxfile)
    getsentence
    getword
    IF NOT EOF(syntaxfile) THEN
      WHILE ch = ""
        getword
      WEND
      LSYSTEM
    END IF
  WEND
FIN

SUB Anweisung
  IF ch = "F" THEN
    semantic21
    getword
  ELSEIF ch = "Fr" THEN
    semantic22
    getword
  ELSEIF ch = "Fl" THEN
    semantic23
    getword
  ELSEIF ch = "f" THEN
    semantic24
    getword
  ELSEIF ch = "X" THEN
    semantic25
    getword
  ELSEIF ch = "r" THEN
    semantic26
    getword
  ELSEIF ch = "l" THEN
    semantic27
    getword
  ELSE
    ERR0R
  END IF
END SUB

SUB Ausdruck
  IF ch = "+" THEN
    semantic7
    getword
    Ausdruck
  ELSEIF ch = "-" THEN
    semantic8
    getword
    Ausdruck
  ELSEIF ch = "[" THEN
    semantic9
    getword
    Befehl
    IF ch = "]" THEN
      semantic10
      getword
    ELSE
      ERR0R
    END IF
  ELSEIF chINfirst("Anweisung") THEN
    Anweisung
  ELSE
    ERR0R
  END IF
END SUB

SUB Befehl
  Ausdruck
  WHILE chINfirst("Ausdruck")
    Ausdruck
  WEND
END SUB

SUB BEGIN
```

```
  semantic = 1
  CLS
  file = COMMAND'$
  path = ""
  OPEN path + file + ".TXT" FOR INPUT AS syntaxfile
 'Here User defined Preparations are added.
  CLS
  SCREEN 12
  VIEW (0, 0)-(639, 479)
  WINDOW (-10, -2 * 3 / 4)-(10, 18 * 3 / 4)
  Length = .1
  Stack = 0
  Status(Stack).x = 0
  Status(Stack).y = 0
  Status(Stack).d = 0
END SUB

FUNCTION chINfirst (char AS STRING)
  chINfirst = FALSE
  SELECT CASE char
    CASE "WHILE"
      IF ch = "|" THEN
        chINfirst = TRUE
      END IF
    CASE "Words"
      IF NOT (ch = "Symbol" OR ch = "!" OR ch = "|" OR ch = "`" OR ch = "'" OR ch = "[" OR ch =
"]" OR ch = "{" OR ch = "}" OR ch = ".") THEN
        chINfirst = TRUE
      END IF
    CASE "Expression"
      IF ch = "Words" OR chINfirst("Words") OR ch = "Symbol" OR ch = "|" OR ch = "`" OR ch = "["
OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Term"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Factor"
      IF chINfirst("Words") OR ch = "Symbol" OR ch = "`" OR ch = "[" OR ch = "{" THEN
        chINfirst = TRUE
      END IF
    CASE "Deklaration"
      IF ch = "d=" THEN
        chINfirst = TRUE
      END IF
    CASE "Produktion"
      IF ch = "F" THEN
        chINfirst = TRUE
      END IF
    CASE "Befehl"
      IF chINfirst("Ausdruck") THEN
```

```
        chINfirst = TRUE
      END IF
    CASE "Ausdruck"
      IF ch = "[" OR ch = "+" OR ch = "-" OR chINfirst("Anweisung") THEN
        chINfirst = TRUE
      END IF
    CASE "Ziffer"
      IF ch = "0" OR ch = "1" OR ch = "2" OR ch = "3" OR ch = "4" OR ch = "5" OR ch = "6" OR ch =
"7" OR ch = "8" OR ch = "9" THEN
        chINfirst = TRUE
      END IF
    CASE "Anweisung"
      IF ch = "F" OR ch = "Fr" OR ch = "Fl" OR ch = "f" OR ch = "X" OR ch = "r" OR ch = "l" THEN
        chINfirst = TRUE
      END IF
    CASE "etc."  'Add user specific conditions
      IF ch = "first('etc.')" THEN
        chINfirst = TRUE
      END IF
  END SELECT
END FUNCTION

SUB Deklaration
  IF ch = "d=" THEN
    semantic4
    getword
  ELSE
    ERR0R
  END IF
  Ziffer
  Ziffer
  IF ch = "." THEN
    semantic5
    getword
  ELSE
    ERR0R
  END IF
  Ziffer
END SUB

SUB ERR0R
  PRINT "      "; CHR$(27); "SYNTAX ERROR!"
  PRINT
  END
END SUB

SUB FIN
 'Here User defined Operations are added.
  CLOSE #syntaxfile
  END
END SUB
```

```
SUB getsentence
  LINE INPUT #syntaxfile, sentence
  PRINT
END SUB

SUB getword
  lastch = ch
  ch = nextch
  IF INSTR(sentence, " ") = 0 THEN
    cha$ = sentence
    sentence = ""
  ELSE
    cha$ = LEFT$(sentence, INSTR(sentence, " ") - 1)
    sentence = RIGHT$(sentence, LEN(sentence) - INSTR(sentence, " "))
  END IF
  nextch = cha$
 'PRINT ch;
END SUB

SUB LSYSTEM
  Deklaration
  IF ch = "," THEN
    semantic2
    getword
  ELSE
    ERR0R
  END IF
  Produktion
  IF ch = ";" THEN
    semantic3
    getword
  ELSE
    ERR0R
  END IF
END SUB

SUB Produktion
  IF ch = "F" THEN
    semantic6
    getword
  ELSE
    ERR0R
  END IF
  WHILE chINfirst("Ausdruck")
    Ausdruck
  WEND
END SUB

SUB Semantic0
 'Here the semantic of "Symbol" is defined.
END SUB
```

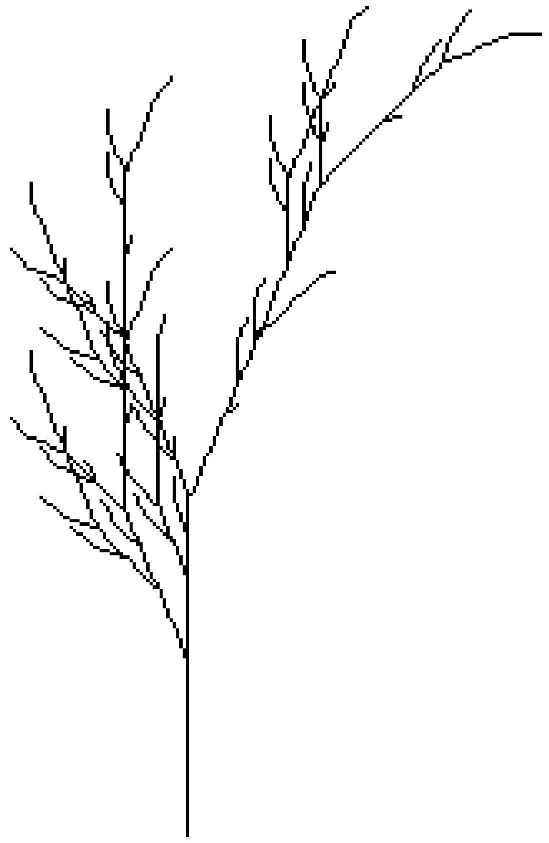
```
SUB Semantic1
 'Here the semantic of "Words" is defined.
END SUB

SUB semantic10
 'Here the semantic of "]" is defined.
  Stack = Stack - 1
END SUB

SUB semantic11
 'Here the semantic of "0" is defined.
  Number = Number + "0"
END SUB

SUB semantic12
 'Here the semantic of "1" is defined.
  Number = Number + "1"
END SUB

SUB semantic13
 'Here the semantic of "2" is defined.
  Number = Number + "2"
END SUB

SUB semantic14
 'Here the semantic of "3" is defined.
  Number = Number + "3"
END SUB

SUB semantic15
 'Here the semantic of "4" is defined.
  Number = Number + "4"
END SUB

SUB semantic16
 'Here the semantic of "5" is defined.
  Number = Number + "5"
END SUB

SUB semantic17
 'Here the semantic of "6" is defined.
  Number = Number + "6"
END SUB

SUB semantic18
 'Here the semantic of "7" is defined.
  Number = Number + "7"
END SUB

SUB semantic19
 'Here the semantic of "8" is defined.
  Number = Number + "8"
```

```
END SUB

SUB semantic2
 'Here the semantic of "," is defined.
  Delta = VAL(Number)
END SUB

SUB semantic20
 'Here the semantic of "9" is defined.
  Number = Number + "9"
END SUB

SUB semantic21
 'Here the semantic of "F" is defined.
  semantic6
END SUB

SUB semantic22
 'Here the semantic of "Fr" is defined.
  semantic6
END SUB

SUB semantic23
 'Here the semantic of "Fl" is defined.
  semantic6
END SUB

SUB semantic24
 'Here the semantic of "f" is defined.
  OldStatus = Status(Stack)
  Status(Stack).x = Status(Stack).x + Length * SIN(Status(Stack).d * RAD)
  Status(Stack).y = Status(Stack).y + Length * COS(Status(Stack).d * RAD)
END SUB

SUB semantic25
 'Here the semantic of "X" is defined.
 '"X" is ignored!
END SUB

SUB semantic26
 'Here the semantic of "r" is defined.
 '"r" is ignored!
END SUB

SUB semantic27
 'Here the semantic of "l" is defined.
 '"l" is ignored!
END SUB

SUB semantic3
 'Here the semantic of ";" is defined.
END SUB
```

```
SUB semantic4
 'Here the semantic of "d=" is defined.
  Delta = 0
  Number = ""
END SUB

SUB semantic5
 'Here the semantic of "." is defined.
  Number = Number + "."
END SUB

SUB semantic6
 'Here the semantic of "F" is defined.
  OldStatus = Status(Stack)
  Status(Stack).x = Status(Stack).x + Length * SIN(Status(Stack).d * RAD)
  Status(Stack).y = Status(Stack).y + Length * COS(Status(Stack).d * RAD)
  LINE (OldStatus.x, OldStatus.y)-(Status(Stack).x, Status(Stack).y), 15
END SUB

SUB semantic7
 'Here the semantic of "+" is defined.
  OldStatus = Status(Stack)
  Status(Stack).d = Status(Stack).d + Delta
END SUB

SUB semantic8
 'Here the semantic of "-" is defined.
  OldStatus = Status(Stack)
  Status(Stack).d = Status(Stack).d - Delta
END SUB

SUB semantic9
 'Here the semantic of "[" is defined.
  Stack = Stack + 1
  Status(Stack) = Status(Stack - 1)
END SUB

SUB Symbol
  Semantic0
  getword
END SUB

SUB Words
 'Here the names of the rules of the language are parsed.
  Semantic1
  getword
END SUB

SUB Ziffer
  IF ch = "0" THEN
    semantic11
    getword
  ELSEIF ch = "1" THEN
    semantic12
    getword
  ELSEIF ch = "2" THEN
    semantic13
    getword
  ELSEIF ch = "3" THEN
    semantic14
    getword
  ELSEIF ch = "4" THEN
    semantic15
    getword
  ELSEIF ch = "5" THEN
    semantic16
    getword
  ELSEIF ch = "6" THEN
    semantic17
    getword
  ELSEIF ch = "7" THEN
    semantic18
    getword
  ELSEIF ch = "8" THEN
    semantic19
    getword
  ELSEIF ch = "9" THEN
    semantic20
    getword
  ELSE
    ERR0R
  END IF
END SUB
==============
```

Als Ausgabe zur Laufzeit erzeugt dieser Compiler Bilder der folgenden Art,wenn er z.B. einen Satz wie oben [LSYSTEM.TXT] liest:

```
==============
[C_LSYSTE.PCX]
```



1

=============

Damit sind wir in der Lage, für jede beliebige formale Sprache, die sich auf die gezeigte Art beschreiben läßt, einen Compiler zu erzeugen.

# LITERATUR

[Wirth]Compilerbau: e. Einf./von Nicklaus Wirth.-
2., durchges. Aufl.- Stuttgart: Teubner, 1981
(Leitfäden der angewandten Mathematik und Mechanik; Bd. 36) (Teubner-Studienbücher: Informatik)
ISBN 3-519-12338-X

[Hückstädt] QBasic/Jürgen Hückstädt.
Hrsg. von Josef Steiner und Robert Valentin.-
Haar bei München: Markt-und-Technik-Verl., 1991
(Schnellübersicht)
ISBN 3-87791-252-4

[Lindenmayer] The Algorithmic Beauty of Plants/by Przemyslaw Prusinkiewicz and Aristid Lindenmayer.
With James S. Hanan, F. David Fracchia, Deborah R. Fowler, Martin J. M. de Boer, Lynn Mercer.-
Berlin: Springer-Verlag, 1990
ISBN 3-540-997297-8