A SHORT DESCRIPTION OF A TRANSLATOR WRITING SYSTEM

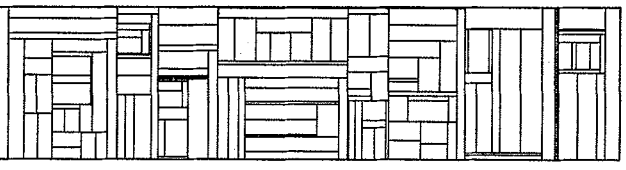
(BOBS - SYSTEM)

by

Bent Bruun Kristensen
Ole Lehrmann Madsen
Bent Bæk Jensen
Søren Henrik Eriksen

DAIMI PB-41
October 1974

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06-12 83 55



A SHORT DESCRIPTION OF A TRANSLATOR WRITING SYSTEM

(BOBS - SYSTEM)

BY

BENT BRUUN KRISTENSEN
OLE LEHRMANN MADSEN
BENT BÆK JENSEN
SØREN HENRIK ERIKSEN

DEPARTMENT OF COMPUTER SCIENCE
INSTITUTE OF MATEMATICS

UNIVERSITY OF AARHUS
DENMARK

ABSTRACT

This paper is itself an abstract describing a translator-writing-system called the BOBS-SYSTEM, which is an implementation of some of the ideas in the Ph.D. thesis of De Remer.

De Remer's thesis describes closely the parsing techniques for the hierachy of LR(K)-grammars. The main parts of the BOBS-SYSTEM are described without many details. Appendix A is a short user manual for the system.

This paper is a revised edition of DAIMI PB-11.

0. CONTENTS

1. INTRODUCTION

This paper is a short description of a translator writing system called the BOBS-SYSTEM. It is an implementation of some of the ideas in the Ph.D. thesis of De Remer [1]. The class of grammars in consideration is the LR(K)-grammars, first described by Knuth in [3], whose implementation requires very large tables. However De Remer claims that by using his techniques one achieves parsers, which are competive in both space and time with precedence parsers. Horning and Lalonde discuss this topic closer in [5]. De Remer defines a hierarchy of LR(K)-grammars in ascending order of complexity by LR(0), SLR(K), LALR(K), L(M)R(K) and LR(K). What we have done is to implement a parsergenerator in the programming language Pascal [6] for the SLR(1)- and LALR(1)-grammars.

2. NOTATION

The reader has to be familiar with finite state machines (FSM's), deterministic push down automata (DPDA's), and context free grammars. An inadequate state is a state in the FSM where applying a production is inconsistent with applying other productions or reading symbols at the same time.

3. GLOBAL DESIGN OF THE SYSTEM

First of all the system is divided into two parts:

1) the parser-generator
2) the parser

3.1. THE PARSER-GENERATOR

In further detail you can divide the parser-generator in the following parts:

1) input of source grammar
2) grammarchecks
3) generation of the LR(0)-machine

4) Extension to SLR(1)/LALR(1) through look-ahead
5) Optimization according to De Remer
6) Conversion to DPDA (deterministic push-down automata)
7) Further optimizations

3.1.1. INPUT OF SOURCE GRAMMAR

The source grammar has to be written in a slightly modified BNF (Backus Normal Form).

3.1.2. GRAMMARCHECKS

This part can be used as an independent part of the system. If you are designing a grammar for a language it has turned out, that the implemented grammarchecks are very helpful . But of course the grammarchecks are an important part of the system as a whole, because you cannot produce the LR(0)-machine for an ambiguous grammar.

The following is a description of the grammarchecks that the system performs:

1) Left and right recursion.

   The systems checks, whether any nonterminal is both left and right recursive. If so, the grammar is ambigious.

2) Termination.

   The system checks, that all nonterminals can produce a string of only terminals.

3) Erasure.

   The system checks,whether any nonterminal can produce the empty string. If so the grammar is modified so it cannot. (The modified grammar produces the same language).

4) Identical productions.

   The grammar is modified by removing the needless productions.

5) Unused productions.

The system checks, that every nonterminal except the goalsymbol appears in both left and right side of a production.

6) Removing simple productions.

A simple production is a production, of which left and right side consists of only a single nonterminal, and the left side nonterminal does not appear on the left side of any other production. The grammar is modified by eliminating all the simple productions.

7) Connection.

The system checks, that all nonterminals can be derived from the goalsymbol.

3.1.3. GENERATION OF THE LR(0) AND EXTENSION TO SLR(1)/ LALR(1)

The LR(0)-machine is derived by using the technique developed by De Remer in [1] and [2]. If the LR(0) is generated without any inadequate states, then the source grammar is LR(0). On the contrary, if inadequate states exist, you have to repair the machine by making look-ahead. In our case we have implemented the global one look-ahead (SLR(1)) and the local one look-ahead (LALR(1)) and if this is not sufficient one has to change the source grammar.

3.1.4. OPTIMIZATION (5, 6 and 7).

If the source grammar happened to be LR(0), SLR(1) or LALR(1), you will get a rather big machine in both space and time.

The most important optimizations suggested by De Remer and Lalonde [4] are therefore performed at this point.

3.1.5. OUTPUT FROM THE PARSER-GENERATOR

1) A list of the source grammar, exactly as you have written it. Any error according to the syntax of input is marked.

2) The results of the earlier mentioned grammarchecks.

3) The grammar written ina nice BNF with possible modifications.

4) Description of states which are not SLR(1)/LALR(1).

5) An error message table for use when parsing a string in error.

6) The parsetables (the optimized machine) in the form of a selfcontained Pascal-program.

If there are errors in input you will only get 1. Logical errors in the grammar will give you 1, 2 and 3. If there are no logical errors and the grammar is not SLR(1)/LALR(1) you will also get 4. In the case that the grammar is SLR(1)/LALR(1) you will get 1, 2, 3, 5 and 6.

In addition there exist several other output facilities, mentioned in [9] but they are only of little interest for this paper.

3.2. THE PARSER

As mentioned the parser is a pascal-program, which without changes will check the syntax of an input-string written in the language defined by the source-grammar. If you want to add semantic actions this is possible.

The parser is roughly divided into three parts:

1) Lexical analysis
2) Syntax analysis
3) Error-recovery

3.2.1. LEXICAL ANALYSIS

Because of the important role of an effective lexical analysis for the parser several are designed. Two are to be mentioned here. The first is a fairly general but simple one which only transforms the sourceinput into a string of internal values. The second is of greater importance in our point of wiew, since it collects identifiers, constants and strings (defined in a Pascal-like manner), which makes the tables smaller and the parsing faster.

3.2.2. SYNTAX ANALYSIS

The syntax analyser uses the machine produced by the generator, to parse the input-string. When the syntax analyser makes a reduction a procedure 'CODE' is called with the number of the production (reduction) as a parameter. The user may then decide, what kind of semantic action, he wants to perform. This is done by writing the body of the procedure 'CODE', which is the only place one has to change the program but of course you may also add new procedures and declarations.

3.2.3. ERROR-RECOVERY

Discovering an error under parsing, an error-recovery algorithm is called, which in a Pascal-like mannermarks the error-symbol (with marks under error-symbols and matching numbers in the margin), and try to recover the error and continue parsing.

4. EVALUATION

The system has been used in a variety of different projects. A compiler for the language PASCAL [6,7] has been based on the system. The PASCAL grammar consists of more than 250 productions and the generated LALR(1) tables occupie about 1100, 60 bit words on a CDC 6400. Compared with the Zurich Pascal Compiler (version 6.Sept. 72) the needs for core and execution time are nearly the same.

Inside our department the system has been used for various compilers and assemblers. Also it is used in a compiler course, at which the stu-

dents have to write a small compiler. At the Danish Data Archieves* the system has been used to implement special purpose languages, which are designed to ease for example the use of libraries of statistical programs, the handling of files and the controlling of data bases. It is of great advantage to be able to experiment with such langauges.

We conclude that the system is usable in practice and that experience has shown that it is easy to modify grammars to become SLR(1) or LALR(1).

The work was started as an undergraduate project under the guidance of Mr. P. Kornerup whose good ideas and great interest have been of great help to the accomplishment of the project.

5. FURTHER PROJECTS

As the work has been moving along new projects have arisen. Automatic error recovery in LR-parsing [8] has been studied. For the time being the problem of defining semantics is studied. One project is to extend the system with the Oxford semantic [10]. Another project is to extend the system with facilities for handling symbol-tables, type-checking etc. as used in practical compiler-writing.

Unfortunately the implementation of the system depends on the PASCAL implementation (version 6. Sept. 72) available on CDC 6400. However a standing project is a new implementation of the system to make it completely portable.

---

*An institution under the Danish Social Science Research Council

6. REFERENCES

[1] De Remer, F,L.
"Practical Translation for LR(K) Languages".
PH. D. Thesis, Massachusetts Institute of Technology,
Cambridge, Mass. August 1969.

[2] De Remer, F,L.
"Simple LR(K) Grammars".
CACM p. 453-459. (14,7,1971)

[3] Knuth,D. E.
"On the Translation of Languages from Left to Right".
INF. and CONT. p. 607-639. (oct. 1965).

[4] Lalonde
"An efficient LALR-Parser-Generator".
Tech. Report CSRG-2, University of Toronto.
Toronto, Ontario, 1971.

[5] Horning, J. J.
Lalonde, W. R.
"Empirical Comparison of LR(K) and precedence Parsers".
Tech. Report CSRG-1. University of Toronto.
Toronto, Ontario, 1970.

[6] Wirth, N.
"The Programming Language Pascal"
ACTA informatica 1,35-63(1971).

[7] Bent. B. Kristensen, Ole L. Madsen, Bent B. Jensen
"An Implementation of a Pascal Compiler"
Unpublished paper, April 1974
Daimi, University of Aarhus

[8] Bent B. Kristensen
"Erkendelse og korrektion af syntaks fejl under LR-parsning"
Master Thesis in danish, May 1974.
Daimi, Aarhus Universitet

[9] Bent B. Jensen, Ole L. Madsen, Bent B. Kristensen,
Søren H. Eriksen
"BOBS-SYSTEM brugervejledning (In Danish)"
Daimi Pb. No. 10, December 1972
Aarhus Universitet

[10] P. Mosses
"The Mathematical Semantics and Compiler Generation"
Ph. D. Thesis, In preparation September 1974
Oxford University Computing Laboratory
Oxford, England.

APPENDIX A

USER MANUAL FOR THE BOBS-SYSTEM.

This paper is an abbreviated, but complete (hopefully) English version of the BOBS-system uder manual (BOBS-SYSTEM, BRUGERVEJLEDNING, DAIMI pb. no. 10 and 22).

A.1 NOTATION

<A> metavariable

[A B C] at most one of the clauses A, B or C may occur (i.e. optional clause)

{A B C} precise one of the clauses A, B or C must occur

... the preceding clause may be repeated zero or more times.

A.2 SYNTAX OF INPUT TO THE PARSER GENERATOR.

```
[OPTIONS ( <OPTION-NUMBER> [, <OPTION-NUMBER>]...)]
[<METASYMBOL-DEFINITIONS>]
{<TERMINAL>... <M4>}
[STRINGCH=<CH> <M4>]
[GOALSYMBOL=<NONTERMINAL> <M4>]
{ <GRAMMAR-RULE> [<METASYMBOL-DEFINITIONS>]}...
{<M4>}
```

<OPTION-NUMBER>

is a integer from 1 to 30. Most of the options are for test purposes and some may cause an error in the generated parser. The most useful ones are: 1, 6, 8, 9, 10, 11, 26, 27, 30.

1. Internal values of all terminals are printed.
6. Internal values of all nonterminals are printed.
8. No listning of input to the generator.
9. No output from grammar checks.
10. No listning of grammar in BNF.
11. No listning of error message table.
26. Extended parser with NAME, KONST and STRING.
27. LALR(1) lookahead instead of SLR(1) lookahead.
30. The LR(0) machine is printed.

<METASYMBOL-DEFINITIONS>

METASYMBOLS M1=<CH> M2=<CH> M3=<CH> M4=<CH>

Correspond to the following symbols in BNF:

M1 is the same as ::=
M2 is the same as |
M3 is the same as < and >
M4 indicates the termination of a sequence of alternatives in a grammar rule.

M1, M2, M3, M4 must all be different.
Default metasymbols are:
M1== M2=/ M3=< M4=;

<TERMINAL>

All terminal symbols used in the grammar must be listed. A terminal symbol consists of a most 10 characters. The character set has been divided into two classes:

1. Letters and digits
2. All other characters except space and end-of line(eol)

All the characters forming a terminal must belong to the same set of the above groups. Terminals consisting of symbols from group 1 must start with a letter. The terminal symbols in the list must be delimited by spaces and/or end-of lines.

The following terminals have a special interpretation. If they are used in the grammar, they must be listed among the other terminals.

EMPTY denotes the empty string.

NAME an identifier is legal in this place of the grammar. (a sequence of letters and digits, with the first symbol being a letter).

KONST a constant is legal in this place of the grammar. (a sequence of letters and digits, with the first symbol being a digit).

STRING a string is legal in this place of the grammar. A string is a sequence of characters surrounded by a string-escape-character (see later). If the string-escape-character is used in the string, it must be written two times per occurence.

STRINGCH=<CH> <M4>

Defines the string-escape-character to be the character <CH>. It must not be part of any other terminal symbol. No default value exists.

GOALSYMBOL=<NONTERMINAL> <M4>

Defines the nonterminal to be the goalsymbol of the grammar. If not present the first nonterminal met in the grammar is assumed to be the goalsymbol. The generator always adds the following grammar rule (production no. 0):

<BOBS-GOAL> ::= <GOALSYMBOL> end-of-file

<GRAMMAR-RULE>

<M3> <NONTERMINAL> <M3> <M1>
<ALTERNATIVE> [<M2> <ALTERNATIVE>]...<M4>

<ALTERNATIVE>

{ <M3> <NONTERMINAL> <M3> / <TERMINAL> }...

The terminals in a grammar rule may not contain any of the metasymbols currently defined. If they do, the metasymbols must be redefined. The terminals in the list must be delimited by spaces and/or end-of lines.

<M1>, <M2>, <M3>, <M4>

Denotes the metasymbols defined by the last metasymbol-definition statement.

<NONTERMINAL>

A sequence of characters not containing the current <M3>. Spaces and end-of-lines are skipped.

<CH>

Any character except letters, digits, space and end-of-line.

A. 3 ERROR MESSAGES.

Two types of error messages can occur:

1. Error messages according to syntax errors in the input to the parser generator. If an error is met in a grammar rule, the message is printed after the next <M4>.

2. Errors caused by table overflow in the parsergenerator. The appropriate constant must be changed in the parser generator.

A. 4 THE PARSER.

NOTE: option 26 must be set when using the parser described in this paper.
When using the parser generator, the parser must reside on the local file PARSIN. The parser generator delivers the parser with initialized tables on the file PARSOUT. The user can extend this program with semantic procedures. New variable declarations are not allowed before the comment:
↱→-end-of-parser-variables-↓

The terminal symbols in the string to be parsed must be delimited by spaces or end-of-lines. However two terminals may be concatenated if they are not in the same group of characters (see <TERMINAL>). Terminals from group2 may be concatenated if they do not together form the head of another terminal.

If NAME, KONST, and STRING are used one can get access to the last sequence of characters which has formed a NAME, a KONST or a STRING.

The array NAMECH contains the last scanned NAME from cell1 to cell NAMENO.

The array KONST contains the last scanned KONST from cell 1 to cell KONSTNO.

The array STRING contains the last scanned STRING from cell1 to cell STRINGNO.

NAMENO, KONSTNO, STRINGNO are integers variables.
Note that the grammar must be formed so the semantic procedures can do something to the above arrays before they are overwritten by the next occurences of a NAME, KONST or STRING on input. However in very special cases this may not be possible. A warning message is then given by the generator and the grammar should be modified.

A. 5 ERROR RECOVERY.

If an error is detected by the parser, the involved symbols are marked with "↑", "-" or "+". "↑" means that the symbol is illegal in this place. "-" means the same, but the symbol has been deleted by the error recovery routine. "+" means that the symbol has been inserted before the preceding symbol.

Example:

GRAMMAR:

<EXPRESSION>::= <EXPRESSION> + <TERM> I <TERM>
<TERM>::= <PRIMARY> * <TERM> I <PRIMARY>
<PRIMARY>::= NAME I (<EXPRESSION>)

INPUT TO THE PARSER:

(AB-CDE)))+(*FG)

OUTPUT FROM THE PARSER:

```
(AB-CDE + ))) +(* NAME FG)                    2   0   3   0   1   0
   -     +  --  ↑     +
```

INTERPRETATION OF OUTPUT FROM THE PARSER:

(AB+CDE) + (NAME*FG)

As can be seen, + and NAME is inserted.

A.6 EXAMPLE

card deck :

```
DATZZ,CM47000.
RFL,2000.
ATTACH,BOBS,BOBSSYSTEM,ID=DATZZ.
ATTACH,PARSIN,BOBSSYSTEM,ID=DATZZ,CY=3.
RFL,100.
RFL,47000.
PASCAL,LOAD=BOBS.
REWIND,PARSOUT.
PASCAL,P=PARSOUT,L=NIL.
```

```
  OPTIONS(26)
 METASYMBOLS M1=↱ M2=$ M3=Ξ M4=↓
 DECLARE IF THEN ELSE FI WHILE DO OD
 READ WRITE ( ) EOL [ ] < ≤ = ≠ ≥ > + - / *
 . ; , : := EMPTY KONST NAME ↓
  ΞPROGRAMΞ ↱ ΞDECLARATIONΞ ΞSTATEMENT-SEQΞ. ↓
 ΞDECLARATIONΞ ↱ DECLARE  ΞVARLISTΞ ;
            $ EMPTY ↓
 ΞVARLISTΞ ↱ ΞVARLISTΞ , ΞITEMΞ  $  ΞITEMΞ  ↓
 ΞITEMΞ ↱ ΞIDΞ  $  ΞIDΞ [ ΞCONSTANTΞ : ΞCONSTANTΞ ] ↓
  ΞSTATEMENT-SEQΞ ↱ ΞSTATEMENT-SEQΞ ; ΞSTATEMENTΞ  $  ΞSTATEMENTΞ  ↓
  ΞSTATEMENTΞ ↱ EMPTY
     $ ΞVARIABLEΞ  := ΞEXPΞ
    $ IF ΞEXPΞ THEN ΞSTATEMENT-SEQΞ ELSE ΞSTATEMENT-SEQΞ FI
     $ WHILE ΞEXPΞ DO ΞSTATEMENT-SEQΞ OD
   $ IF ΞEXPΞ THEN ΞSTATEMENT-SEQΞ FI
   $ READ( ΞVARIABLEΞ )  $  WRITE( ΞEXPΞ ) $ EOL ↓
   ΞEXPΞ ↱ ΞAEXPΞ ΞRELOPΞ ΞAEXPΞ  $  ΞAEXPΞ ↓
  ΞAEXPΞ ↱ ΞAEXPΞ ΞADDOPΞ ΞTERMΞ $ ΞTERMΞ ↓
  ΞTERMΞ ↱ ΞTERMΞ ΞMULTOPΞ ΞPRIMARYΞ $ ΞPRIMARYΞ ↓
  ΞPRIMARYΞ ↱ ΞVARIABLEΞ $ ΞCONSTANTΞ $ ( ΞEXPΞ ) ↓
 ΞVARIABLEΞ ↱ ΞIDΞ  $ ΞIDΞ [ ΞEXPΞ ] ↓
 ΞRELOPΞ ↱ < $ ≤ $ ≠ $ = $ > $ ≥ ↓
 ΞADDOPΞ ↱ + $ - ↓
 ΞMULTOPΞ ↱ * $ / ↓
 ΞIDΞ ↱ NAME ↓
 ΞCONSTANTΞ ↱ KONST $ + KONST $ - KONST ↓
 ↓
DECLARE N,UB,LB,I,T,BOO,SUM[0:2];
   READ(N); READ(UB); READ(LB);
   I:=1;
   WHILE I≤N DO
      READ(T);
      IF 0≤T THEN
         BOO:=(LB≤T)*(T≤UB);
         SUM[BOO]:=SUM[BOO]+BOO*T*I;
         SUM[BOO]:=SUM[BOO]+(1-BOO)*T/I
      FI;
      I:=I+1
   OD;
   SUM[1+1]:=((SUM[0]+SUM[1])*(UB-LB)/(UB+LB))/2;
   WRITE(SUM[2])  .
```

```
OPTIONS(26)

METASYMBOLS M1=↱ M2=$ M3=Ξ M4=↓

          DECLARE IF THEN ELSE FI WHILE DO OD
 READ WRITE ( ) EOL [ ] < ≤ = ≠ ≥ > + - / *
 . ; , : := EMPTY KONST NAME ↓

           ΞPROGRAMΞ ↱ ΞDECLARATIONΞ ΞSTATEMENT-SEQΞ. ↓

          ΞDECLARATIONΞ ↱ DECLARE  ΞVARLISTΞ ;

                    $ EMPTY ↓

          ΞVARLISTΞ ↱ ΞVARLISTΞ , ΞITEMΞ  $  ΞITEMΞ  ↓

          ΞITEMΞ ↱ ΞIDΞ  $  ΞIDΞ [ ΞCONSTANTΞ : ΞCONSTANTΞ ] ↓

           ΞSTATEMENT-SEQΞ ↱ ΞSTATEMENT-SEQΞ ; ΞSTATEMENTΞ  $  ΞSTATEMENTΞ  ↓

           ΞSTATEMENTΞ ↱ EMPTY

              $ ΞVARIABLEΞ  := ΞEXPΞ

             $ IF ΞEXPΞ THEN ΞSTATEMENT-SEQΞ ELSE ΞSTATEMENT-SEQΞ FI

              $ WHILE ΞEXPΞ DO ΞSTATEMENT-SEQΞ OD

            $ IF ΞEXPΞ THEN ΞSTATEMENT-SEQΞ FI

            $ READ( ΞVARIABLEΞ )  $  WRITE( ΞEXPΞ ) $ EOL ↓

            ΞEXPΞ ↱ ΞAEXPΞ ΞRELOPΞ ΞAEXPΞ  $  ΞAEXPΞ ↓

           ΞAEXPΞ ↱ ΞAEXPΞ ΞADDOPΞ ΞTERMΞ $ ΞTERMΞ ↓

           ΞTERMΞ ↱ ΞTERMΞ ΞMULTOPΞ ΞPRIMARYΞ $ ΞPRIMARYΞ ↓

           ΞPRIMARYΞ ↱ ΞVARIABLEΞ $ ΞCONSTANTΞ $ ( ΞEXPΞ ) ↓

          ΞVARIABLEΞ ↱ ΞIDΞ  $ ΞIDΞ [ ΞEXPΞ ] ↓

          ΞRELOPΞ ↱ < $ ≤ $ ≠ $ = $ > $ ≥ ↓

          ΞADDOPΞ ↱ + $ - ↓

          ΞMULTOPΞ ↱ * $ / ↓

          ΞIDΞ ↱ NAME ↓

          ΞCONSTANTΞ ↱ KONST $ + KONST $ - KONST ↓

          ↓
```

********** GRAMMARCHECKS **********

IT HAS BEEN CHECKED THAT ALL NONTERMINALS
EXCEPT THE GOALSYMBOL APPEAR IN BOTH
LEFT AND RIGHTSIDE OF A PRODUCTION

IT HAS BEEN CHECKED THAT THERE
EXISTS NO IDENTICAL PRODUCTIONS

THE GRAMMER HAS BEEN CHECKED FOR
SIMPLE CHAINS

IT HAS BEEN CHECKED THAT ALL NONTERMINALS CAN
PRODUCE A STRING OF ONLY TERMINAL SYMBOLS

THE GRAMMAR IS MODIFIED FOR ERASURE
I. E. NO NONTERMINAL CAN NOW PRODUCE THE EMPTY STRING

IT HAS BEEN CHECKED THAT NO NONTERMINAL
IS BOTH LEFT AND RIGHT RECURSIVE

```
 1   <PROGRAM> ::= <DECLARATION> <STATEMENT-SEQ> .
 2               / <DECLARATION> .
 3               / <STATEMENT-SEQ> .
 4               / .

 5   <DECLARATION> ::= DECLARE <VARLIST> ;

 6   <STATEMENT-SEQ> ::= <STATEMENT-SEQ> ; <STATEMENT>
 7                     / ; <STATEMENT>
 8                     / <STATEMENT-SEQ> ;
 9                     / ;
10                     / <STATEMENT>

11   <VARLIST> ::= <VARLIST> , <ITEM>
12               / <ITEM>

13   <ITEM> ::= <ID>
14            / <ID> [ <CONSTANT> : <CONSTANT> ]

15   <ID> ::= NAME

16   <CONSTANT> ::= KONST
17                / + KONST
18                / - KONST

19   <STATEMENT> ::= <VARIABLE> := <EXP>
20                 / IF <EXP> THEN <STATEMENT-SEQ> ELSE <STATEMENT-SEQ> FI
21                 / IF <EXP> THEN <STATEMENT-SEQ> ELSE FI
22                 / IF <EXP> THEN ELSE FI
23                 / IF <EXP> THEN ELSE <STATEMENT-SEQ> FI
24                 / WHILE <EXP> DO <STATEMENT-SEQ> OD
25                 / WHILE <EXP> DO OD
26                 / IF <EXP> THEN <STATEMENT-SEQ> FI
27                 / IF <EXP> THEN FI
28                 / READ ( <VARIABLE> )
29                 / WRITE ( <EXP> )
30                 / EOL

31   <VARIABLE> ::= <ID>
32                / <ID> [ <EXP> ]

33   <EXP> ::= <AEXP> <RELOP> <AEXP>
34           / <AEXP>

35   <AEXP> ::= <AEXP> <ADDOP> <TERM>
36             / <TERM>

37   <RELOP> ::= <
38              / ≤
39              / ≠
40              / =
41              / >
42              / ≥

43   <ADDOP> ::= +
44              / -

45   <TERM> ::= <TERM> <MULTOP> <PRIMARY>
46             / <PRIMARY>

47   <MULTOP> ::= *
48               / /

49   <PRIMARY> ::= <VARIABLE>
50                / <CONSTANT>
51                / ( <EXP> )
```

****

THE GRAMMAR IS SLR1

****

********** COMPILER ERROR MESSAGES **********

ERRORNO : 0 ** SPECIAL ERROR **

ERRORNO	EXPECTED SYMBOL:				
1	.	DECLARE	;	IF	WHILE
	READ	WRITE	EOL	NAME	
2	(	KONST	+	-	NAME
3	KONST				
4	)				
5	]				
6	(				
7	NAME				
8	DO				
9	OD	;	IF	WHILE	READ
	WRITE	EOL	NAME		
10	THEN				
11	ELSE	FI	;	IF	WHILE
	READ	WRITE	EOL	NAME	
12	:=				
13	FI	;	IF	WHILE	READ
	WRITE	EOL	NAME		
14	FI	;			
15	ELSE	FI	;		
16	OD	;			
17	KONST	+	-		
18	:				
19	;	,			
20	.	;			
21	.	;	IF	WHILE	READ
	WRITE	EOL	NAME		
22	-EOF-				

****

NOTICE: BOBS-PARSER WITH ≠NAME≠ , ≠KONST≠ AND ≠STRING≠ MUST BE USED

Output from the parser

```
DECLARE N,UB,LB,I,T,BOO,SUM[0:2];
   READ(N); READ(UB); READ(LB);
   I:=1;
   WHILE I≤N DO
      READ(T);
      IF 0≤T THEN
         BOO:=(LB≤T)*(T≤UB);
         SUM[BOO]:=SUM[BOO]+BOO*T*I;
         SUM[BOO]:=SUM[BOO]+(1-BOO)*T/I
      FI;
      I:=I+1
   OD;
   SUM[1+1]:=((SUM[0]+SUM[1])*(UB-LB)/(UB+LB))/2;
   WRITE(SUM[2])   .
```

```
COMPILATION-TIME :          0.       137 SEK.
```