CLEMSW User's Manual

by

Geoffrey M. Clemm

Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado  80309

CS-CU-167-79  November, 1979
Abstract

CLEMSW is a modified version of the December 1974 release of BOBSW, an LALR (1) parser generating system. [1].
I CLEMSW input

The input to CLEMSW consists of two sections. Section one specifies the interface to a lexical analyzer produced by the FSCAN system [2] and Section two specifies the language to be parsed. Each section is preceded by a keyword that must begin in column one. All other input is free format but may not be in column one. Any sequence of spaces and/or end-of-line's forms a separator between input items.

Section 1: Keyword = LEXEMES

A scanner for the parser should be specified in the language FSCAN. The tokens specified as being produced by the scanner are the terminals, or lexemes, of the grammar input to CLEMSW.

Following the keyword, LEXEMES, should be a list of terminal names, in the order that the terminals appear in the common block, /TKTPCM/, in the block data subprogram, TKTPBD, produced by the FSCAN compiler.

A terminal name may be a sequence of alphanumeric characters (the first character must be alphabetic) or a sequence of operator characters ('+', '*', '/') or '(', etc.).

Section 2: Keyword = PRODUCTIONS

Following the PRODUCTIONS keyword, the grammar is specified in standard BNF notation. The four metasymbols needed to specify a BNF grammar are:

1. Separator between the right and left side of a production, ":=" in BNF, default is ":&".

2. Separator between alternatives in the right side of a production, ":/" in BNF, default is ":$".

3. Delimiter for the beginning and end of a nonterminal, ":<" and ":>" in BNF, default is ":" (a quote mark).

4. Delimiter for the end of a production, ":;" in BNF, default is ":#".

If metasymbols other than the default metasymbols are desired, they can be specified immediately before the first production by a line starting with the keyword METASYMBOLS, as follows:
METASYMBOLS M1 = A  M2 = B  M3 = C  M4 = D
where A, B, C, and D are distinct characters to be used as metasymbols one, two, three, and four respectively.

The format for terminals is described in Section one. Nonterminals consist of metasymbol three, followed by a sequence of 1 to 30 characters (any character other than metasymbol three), followed by metasymbol three.

The left side of the first production is the goal symbol for the grammar.

Immediately following the metasymbol two or four demarking the end of a particular alternative, is placed an unsigned integer that will be sent as an argument to the semantic routine when that alternative is reduced during parsing. This integer must be less than or equal to the total number of alternatives in the grammar.

To indicate the end of the input to CLEMSW, metasymbol four is placed in column one.
II  Restrictions on Input

To avoid processing erroneous grammars, the following restrictions are placed on input grammars.

1. Each terminal must appear in a production.
2. Each nonterminal except for the goal symbol must appear in the left side of a production and the right side of a production.
3. Each nonterminal must be able to produce a string of only terminals.
4. A nonterminal cannot be both left and right recursive.

In addition, the design of the BOBSW system necessitates the following restrictions.

1. Each alternative must contain at most seven terminals or nonterminals.
2. No empty productions can be used.
III Output from CLEMSW

CLEMSW produces a listing and a tables file. Included on the listing file are any error messages. If the input grammar is not LALR1, all inadequate states are printed. If the grammar is LALR1, tables are produced in the form of a FORTRAN block data subprogram. These tables, when combined with an FSCAN generated scanner and the FORTRAN parser driver (both machine independent FORTRAN programs), form a parser for the input grammar. The semantic actions produced by the parser consist of a call to the FORTRAN subroutine, SUBROUTINE ACTION(I), where the input parameter, I, is the value specified following metasymbol 2 or 4 of the alternative being reduced.
IV Availability of CLEMSW at the University of Colorado

CLMSWB is the name of the compiled version of CLEMSW. CLMSWB (and CLEMSW) are available through UPLOADIT from tape #110506, project = PAIX, user no. = B578.

CLMSWB requires 111K octal words and 82 seconds to process the FORTRAN grammar described in part five.

Example:

GET, CWINPT, CLMSWB
RFL, 111000
CLMSWB, CWINPT, CWLSTNG, CWTABLES

where CWINPT, CWLSTNG, CWTABLES are the input, listing, and tables file, respectively.

The FORTRAN parser driver is obtainable as the "PARSER TABLE DRIVER" section of the FPARS file from the same tape.
V Sample Input to CLEMSW

The following LALR(1) grammar specifies a FORTRAN variant that is a subset of CDC FTN FORTRAN, but which contains standard ANSI FORTRAN. This grammar is based on an earlier grammatical specification of ANSI FORTRAN by Frank DeRemer and Russ Rauhauser.

The terminals (listed following the keyword, LEXEMES) are interpreted as follows:

- **DCONST** - decimal constant.
- **DPCONST** - double precision constant.
- **EOS** - end of statement.
- **FMTFLO** - format field.
- **HCOST** - hollerith constant.
- **LCOST** - logical constant.
- **NAME** - FORTRAN identifier
- **OCOST** - octal constant.
- **RCOST** - real constant.
- **RELOP** - relational operator.

All keywords (e.g., BLOCKDATA, GOTO, etc.), non-relational operators (e.g., +, *, .NOT., etc.) and separators (e.g., =, (, etc.) are represented as they would appear in a FORTRAN program, except that the periods are deleted from the logical operators, and EQUIVALENCE and DOUBLEPRECISION are truncated to EQUIVALENCE and DOUblasPES for respectively.

The extensions to ANSI FORTRAN are:

1. PROGRAM statement.
2. IMPLICIT statement.
3. TYPE statement
4. LEVEL statement.
5. Two branch logical and arithmetic IF statements.
6. Extended READ and WRITE statements.
7. ENCODE and DECODE statements.
8. PUNCH statement.
9. Hollerith constants may appear in arithmetic expressions.
10. Array subscripts may be arbitrary expressions.
"COMPLEX" $1
"LOGICAL" #1
"PROGRAM BODY" & "IMPLICITS" "SDFE BLOCK" "END STMT" $16
"SDFE BLOCK" "END STMT" #15
"SPEC BLOCK" "DATAFOAEXEC BLOCK" $2
"DATAFOAEXEC BLOCK" #1
"DATA BLOCK" "FOAEXEC BLOCK" $2
"FOAEXEC BLOCK" #1
"FOAEXEC BLOCK" "EXEC BLOCK" $2
"FOA BLOCK" $1
"EXEC BLOCK" #1
"IMPLICITS" & "IMPLICIT" $1
"IMPLICITS" "IMPLICIT" #2
"SPEC BLOCK" & "SPECS" $1
"SPECS" "SPEC BLOCK" "SPECS" #2
"SPECIFICATION" $1
"FORMAT STMT" #1
"DATA BLOCK" & "DATA STMT" $1
"DATA BLOCK" "DATAS" #2
"DATAS" & "DATA STMT" $1
"FORMAT STMT" #1
"FOA BLOCK" & "FUNCTION OR ARRAY" $1
"FOA BLOCK" "FOAS" #2
"FOAS" & "FUNCTION OR ARRAY" $1
"DATAS" #1
"EXEC BLOCK" & "EXECUTABLE STMT" $1
"EXEC BLOCK" "EXECS" #2
"EXECS" & "EXECUTABLE STMT" $1
"ARR ASMT STMT" $1
"DATAS" #1
"IMPLICIT" & "LABEL" "IMP" EOS $6
"IMP" EOS #1
"IMP" & IMPLICIT "IMP TYPE LIST" #22
"SPECIFICATION" & "LABEL" "SPEC" EOS $6
"SPEC" EOS #1
"DIMENSION "ARRAY DCLN LIST" $17
"COMMON "BLANK COM" $18
"COMMON "BLANK COM" "COM LIST" $19
"COMMON "COM LIST" $18
"EQUIVALENCE "EQUIVALENCE LIST" $20
"TYPE" "TYPE LIST" $21
"EXTERNAL "NAME LIST" $26
"LEVEL "DCONST", "NAMED VALUE LIST" #36
"ARRAY DCLN LIST" & "ARRAY DCLN" $1
"ARRAY DCLN LIST", "ARRAY DCLN" #2
"ARRAY DCLN" & "NAME" ( "SUBSCR LIST" ) #23
"SUBSCR LIST" & "INTEGER" $11
"INTEGER", "INTEGER" #12
"INTEGER", "INTEGER", "INTEGER" #13
"BLANK COM" & "DCLN LIST" #24
"COM LIST" & "COM BLOCK" $1
"COM LIST" "COM BLOCK" #2
"COM BLOCK" & / / "DCLN LIST" $24
/ "NAME" / "DCLN LIST" #25
"FIELD LIST" & "SLASH GROUP" / #10
FINE FIELD S1
"FIELD LIST" "SEP" "FIELD" #2
"SEP" & , $10
"SLASH GROUP" #10
"FIELD" & "BASIC FIELD" $10
"GROUP" #1
"BASIC FIELD" & "HCONST" S1
"FMTFLD" #1
"GROUP" & "DCONST" "FMT" $32
"FMT" #1
"FUNCTION OR ARRAY" & "LABEL" "FOA" EOS $6
"FOA" EOS #1
"FOA" & "ARRAY REF" = "EXPRESSION" #33
"EXECUTABLE STMT" & "LABEL" "EXEC" EOS $6
"EXEC" EOS #1
"EXEC"
& DO "LABEL" "NAME" = "DO PARAMETERS" $88
IF ( "EXPRESSION" ) "BASIC" $34
IF ( "EXPRESSION" ) "ARR ASMT" $34
IF ( "EXPRESSION" ) "LOG IF LABELS" $34
IF ( "EXPRESSION" ) "ARITH IF LABELS" $41
ENTRY "NAME" $35
"BASIC" #1
"DO PARAMETERS"
& "INTEGER", "INTEGER" $12
"INTEGER", "INTEGER", "INTEGER" #13
"LOG IF LABELS" & "LABEL", "LABEL" #12
"ARR ASMT STMT" & "LABEL" "ARR ASMT" EOS $6
"ARR ASMT" EOS #1
"ARR ASMT"
& "ARRAY REF" = "EXPRESSION" #37
"BASIC"
& "NAME" = "EXPRESSION" $37
ASSIGN "LABEL" TO "NAME" $38
GOTO "LABEL" $39
GOTO ( "LABEL LIST" ), "EXPRESSION" $40
GOTO "NAME" . ( "LABEL LIST" ) $90
GOTO "NAME" ( "LABEL LIST" ) $90
CALL "NAME" ( "EXPRN LIST" ) $43
CALL "NAME" $42
RETURN $44
CONTINUE $45
STOP $46
STOP "OCONST" $47
PAUSE $48
PAUSE "OCONST" $49
REWIND "INTEGER" $50
BACKSPACE "INTEGER" $51
ENDFILE "INTEGER" $52
READ "FORMATTED IOSPEC" $53
READ "UNFORMATTED IOSPEC" $53
WRITE "FORMATTED IOSPEC" $54
WRITE "UNFORMATTED IOSPEC" $54
PRINT "FORMATTED IOSPEC" $54
PUNCH "FORMATTED IOSPEC" $54
ENCODE "CODE SPEC" "IO LIST" $55
DECODE "CODE SPEC" "IO LIST" #56
"ARITH IF LABELS" & "LABEL", "LABEL", "LABEL" #13
"LABEL LIST" & "LABEL LIST*" #11
"LABEL LIST*" & "LABEL" $1
   "LABEL LIST*", "LABEL" #2
"FORMATTED IOSPEC" & "XFORMATTED SPEC" "IO LIST" $2
"FORMATTED SPEC" $1
   "FORMATTED XIOSPEC" #1
"FORMATTED XIOSPEC" & "XFORMATTED XSPEC", "IO LIST" $2
   "FORMATTED XSPEC" #1
"UNFORMATTED IOSPEC" & "UNFORMATTED SPEC" "IO LIST" $2
   "UNFORMATTED SPEC" #1
"XFORMATTED SPEC" & ( "INTEGER", * ) $58
   "FORMATTED SPEC" #1
"FORMATTED SPEC" & ( "INTEGER", "FORM" ) #59
"UNFORMATTED SPEC" & ( "INTEGER" ) #58
"XFORMATTED XSPEC" & * $57
   "FORM" #58
"FORMATTED XSPEC" & "FORM" #58
"FORM" & "LABEL" $1
   "NAME" #1
"CODE SPEC" & ( "INTEGER", "FORM", "NAME" ) #60
"IO LIST" & "IO LIST*" #61
"IO LIST*" & "NAMED VALUE" $1
   ( "NAMED VALUE LIST" ) $1
   ( "ITERATION LIST" ) $1
   "NAMED VALUE", "IO LIST*" $2
   ( "NAMED VALUE LIST" ), "IO LIST*" $2
   ( "ITERATION LIST" ), "IO LIST*" #2
"NAMED VALUE LIST" & "NAMED VALUE", "NAMED VALUE LIST" #2
"ITERATION LIST" & "ITERATION LIST*" #86
"ITERATION LIST*" & "NAMED VALUE", "DO SPECIFICATION" $2
   ( "NAMED VALUE LIST" ), "DO SPECIFICATION" $2
   ( "ITERATION LIST" ), "DO SPECIFICATION" $2
   "NAMED VALUE", "ITERATION LIST*" $2
   ( "NAMED VALUE LIST" ), "ITERATION LIST*" $2
   ( "ITERATION LIST" ), "ITERATION LIST*" #2
"DO SPECIFICATION" & "NAME" = "DO PARAMETERS" #62
"END STMT" & END EOS #10
"EXPRESSION" & "L TERM" $1
   "EXPRESSION" OR "L TERM" #63
"L TERM" & "L FACTOR" $1
   "L TERM" AND "L FACTOR" #64
"L FACTOR" & "L PRIMARY" $1
   NOT "L PRIMARY" #65
"L PRIMARY" & "LCONST" $1
   "RELATIONAL EXPRN" $1
   "ARITH EXPRN" #1
"RELATIONAL EXPRN" & "ARITH EXPRN" "RELOP" "ARITH EXPRN" #66
"ARITH EXPRN" & "A TERM" $1
   + "A TERM" $1
   - "A TERM" $29
   "ARITH EXPRN" + "A TERM" $67
VI Sample Output from CLEMSW

With the FORTRAN grammar from part 5 as input, the CLEMSW system produces the following output:
IT HAS BEEN CHECKED THAT ALL NONTERMINALS EXCEPT THE GOALS SYMBOL(S) APPEAR IN BOTH LEFT AND RIGHTSIDE OF A PRODUCTION

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

IT HAS BEEN CHECKED THAT NO NONTERMINAL CAN PRODUCE THE EMPTY STRING

IT HAS BEEN CHECKED THAT NO NONTERMINAL IS BOTH LEFT AND RIGHT RECURSIVE
*************** THE GRAMMAR BEFORE LRO ***************

100 1  <COMPILATION UNIT> ::= <PROGRAM UNIT>
100 2   / <COMPILATION UNIT> <PROGRAM UNIT>

6  3  <PROGRAM UNIT> ::= <LABEL> <BLOCKDATA>
1  4   / <BLOCKDATA>
91  5   / <PROGRAM BODY>
87  6   / <PROGRAM> <PROGRAM BODY>
6  7   / <LABEL> <SUBPROGRAM>
1  8   / <SUBPROGRAM>

73  9  <LABEL> ::= DCONST

7 10  <BLOCKDATA> ::= BLOCKDATA EOS <BLOCKDATA_STMTS>
8 11   / BLOCKDATA <NAME> EOS <BLOCKDATA_STMTS>

16 12  <PROGRAM BODY> ::= <IMPLICITS> <SDFE BLOCK> <END_STMT>
15 13   / <SDFE BLOCK> <END_STMT>

1 14  <PROGRAM> ::= PROGRAM <NAME> EOS
2 15   / PROGRAM <NAME> ( <FILE LIST> ) EOS

9 16  <SUBPROGRAM> ::= <SUBROUTINE> <PROGRAM BODY>
14 17   / <FUNCTION> <PROGRAM BODY>

2 18  <BLOCKDATA_STMTS> ::= <BLOCKSPECs> <BLOCKDASABS> <END_STMT>
1 19   / <BLOCKSPECs> <END_STMT>
1 20   / <BLOCKDASABS> <END_STMT>
89 21   / <END_STMT>

74 22  <NAME> ::= NAME

1 23  <BLOCKSPECs> ::= <SPECIFICATION>
2 24   / <BLOCKSPECs> <SPECIFICATION>

1 25  <BLOCKDASABS> ::= <DATA_STMT>
2 26   / <BLOCKDASABS> <DATA_STMT>

10 27  <END_STMT> ::= END EOS

6 28  <SPECIFICATION> ::= <LABEL> <SPEC> EOS
1 29   / <SPEC> EOS

6 30  <DATA_STMT> ::= <LABEL> <DATA> EOS
1 31   / <DATA> EOS

11 32  <FILE_LIST> ::= <FILE_LIST*>

1 33  <FILE_LIST*> ::= <FILE>
2 34   / <FILE_LIST*> , <FILE>
<FILE> ::= <NAME>
   / <NAME> = <DCONST>
   / <NAME> = <DCONST> / <DCONST>
   / <NAME> = / <DCONST>
   / <NAME> = <NAME>

<DCONST> ::= DCONST

<SUBROUTINE> ::= SUBROUTINE <NAME> EOS
   / SUBROUTINE <NAME> <PARAMETER LIST> EOS

<FUNCTION> ::= <FUNCTION PREFIX> <NAME> <PARAMETER LIST> EOS

<PARAMETER LIST> ::= ( <NAME LIST> )

<FUNCTION PREFIX> ::= FUNCTION
   / <TYPE> FUNCTION

<NAME LIST> ::= <NAME>
   / <NAME LIST>, <NAME>

<TYPE> ::= <INT>
   / <REAL>
   / <DOUBLEPREC>
   / <COMPLEX>
   / <LOGICAL>

<int> ::= INTEGER

<REAL> ::= REAL

<DOUBLEPREC> ::= DOUBLEPREC

<COMPLEX> ::= COMPLEX

<LOGICAL> ::= LOGICAL

<IMPLICITs> ::= <IMPLICIT>
   / <IMPLICITs> <IMPLICIT>

<SDFE BLOCK> ::= <SPEC BLOCK> <DATAFOAEXEC BLOCK>
   / <DATAFOAEXEC BLOCK>

<SPEC BLOCK> ::= <SPECS>
   / <SPEC BLOCK> <SPECS>

<DataFOAEXEC BLOCK> ::= <DATA BLOCK> <FOAEXEC BLOCK>
   / <FOAEXEC BLOCK>

<Data BLOCK> ::= <DATA STMT>
   / <DATA BLOCK> <DATAS>

<FOAEXEC BLOCK> ::= <FOA BLOCK> <EXEC BLOCK>
   / <FOA BLOCK>
1 71 / <EXEC BLOCK>
1 72 <FOA BLOCK> ::= <FUNCTION OR ARRAY>
2 73 / <FOA BLOCK> <FOAS>
1 74 <EXEC BLOCK> ::= <EXECUTABLE STMT>
2 75 / <EXEC BLOCK> <EXECS>
1 76 <IMPLICIT> ::= <LABEL> <IMP> EOS
1 77 / <IMP> EOS
1 78 <SPECS> ::= <SPECIFICATION>
1 79 / <FORMAT STMT>
6 80 <FORMAT STMT> ::= <LABEL> FORMAT <FMT> EOS
1 81 <DATAS> ::= <DATA STMT>
1 82 / <FORMAT STMT>
6 83 <FUNCTION OR ARRAY> ::= <LABEL> <FOA> EOS
1 84 / <FOA> EOS
1 85 <FOAS> ::= <FUNCTION OR ARRAY>
1 86 / <DATAS>
6 87 <EXECUTABLE STMT> ::= <LABEL> <EXEC> EOS
1 88 / <EXEC> EOS
1 89 <EXECS> ::= <EXECUTABLE STMT>
1 90 / <ARR ASMT STMT>
1 91 / <DATAS>
6 92 <ARR ASMT STMT> ::= <LABEL> <ARR ASMT> EOS
1 93 / <ARR ASMT> EOS
22 94 <IMP> ::= IMPLICIT <IMP TYPE LIST>
1 95 <IMP TYPE LIST> ::= <IMP TYPE>
2 96 / <IMP TYPE LIST>, <IMP TYPE>
17 97 <SPEC> ::= DIMENSION <ARRAY DCLN LIST>
18 98 / COMMON <BLANK COM>
19 99 / COMMON <BLANK COM> <COM LIST>
18 100 / COMMON <COM LIST>
20 101 / EQUIVALENC <EQUIVALENCE LIST>
21 102 / <TYPE> <TYPE LIST>
26 103 / EXTERNAL <NAME LIST>
36 104 / LEVEL <DCONST>, <NAMED VALUE LIST>
1 105 <ARRAY DCLN LIST> ::= <ARRAY DCLN>
2 106 / <ARRAY DCLN LIST>, <ARRAY DCLN>
24 107 <BLANK COM> ::= <DCLN LIST>
1 108  <COM LIST> ::= <COM BLOCK>
2 109     / <COM LIST> <COM BLOCK>

1 110  <EQUIVALENCE LIST> ::= <EQUIV LIST>
2 111     / <EQUIVALENCE LIST> , <EQUIV LIST>

1 112  <TYPE LIST> ::= <DCLN ELEMENT>
2 113     / <TYPE LIST> , <DCLN ELEMENT>

1 114  <NAMED VALUE LIST> ::= <NAMED VALUE>
2 115     / <NAMED VALUE> , <NAMED VALUE LIST>

23 116  <ARRAY DCLN> ::= <NAME> ( <SUBSCR LIST> )

11 117  <SUBSCR LIST> ::= <INTEGER>
12 118     / <INTEGER> , <INTEGER>
13 119     / <INTEGER> , <INTEGER> , <INTEGER>

1 120  <INTEGER> ::= <ICONST>
1 121     / <NAME>

11 122  <DCLN LIST> ::= <DCLN LIST*>

24 123  <COM BLOCK> ::= / / <DCLN LIST>
25 124     / / <NAME> / <DCLN LIST>

1 125  <DCLN LIST*> ::= <COMMON DCLN ELEM>
2 126     / <DCLN LIST*> , <COMMON DCLN ELEM>

1 127  <COMMON DCLN ELEM> ::= <NAME>
1 128     / <COMMON ARRAY>

23 129  <COMMON ARRAY> ::= <NAME> ( <DCONST LIST> )

11 130  <DCONST LIST> ::= <DCONST>
12 131     / <DCONST> , <DCONST>
13 132     / <DCONST> , <DCONST> , <DCONST>

12 133  <EQUIV LIST> ::= ( <DECLARATOR> , <EQUIV DECLARATOR LIST> )

23 134  <DECLARATOR> ::= <NAME> ( <DCONST LIST> )
1 135     / <NAME>

1 136  <EQUIV DECLARATOR LIST> ::= <DECLARATOR>
2 137     / <EQUIV DECLARATOR LIST> , <DECLARATOR>

1 138  <DCLN ELEMENT> ::= <NAME>
1 139     / <ARRAY DCLN>

12 140  <IMP TYPE> ::= <TYPE> ( <NAME> )
13 141     / <TYPE> ( <NAME> - <NAME> )

27 142  <DATA> ::= DATA <DATA PAIR LIST>
1 143  <DATA PAIR LIST> ::= <DATA PAIR> 
2 144 / <DATA PAIR LIST> , <DATA PAIR> 
12 145  <DATA PAIR> ::= <DECLARATOR LIST> / <DATA LIST> / 
11 146  <DECLARATOR LIST> ::= <DECLARATOR LIST*> 
11 147  <DATA LIST> ::= <DATA LIST*> 
1 148  <DECLARATOR LIST*> ::= <DECLARATOR> 
2 149 / <DECLARATOR LIST*> , <DECLARATOR> 
1 150  <DATA LIST*> ::= <DATA ITEMS> 
2 151 / <DATA LIST*> , <DATA ITEMS> 
28 152  <DATA ITEMS> ::= <DCONST> * <DATA ITEM> 
1 153 / <DATA ITEM> 
1 154  <DATA ITEM> ::= <HCONST> 
1 155 / <LCONST> 
1 156 / <DATA NUMBER> 
75 157  <HCONST> ::= HCONST 
77 158  <LCONST> ::= LCONST 
1 159  <DATA NUMBER> ::= <NUMBER> 
1 160 / <COMPLEX CONST> 
1 161 / + <NUMBER> 
29 162 / - <NUMBER> 
1 163  <NUMBER> ::= <ICONST> 
1 164 / <RCONST> 
1 165 / <DPCONST> 
71 166  <COMPLEX CONST> ::= ( <CCONST ELEMENT> , <CCONST ELEMENT> ) 
30 167  <FMT> ::= ( ) 
31 168 / ( <FIELD LIST> ) 
30 169 / ( <SLASH GROUP> ) 
31 170 / ( <SLASH GROUP> <FIELD LIST> ) 
31 171 / ( <FIELD LIST> <SLASH GROUP> ) 
31 172 / ( <SLASH GROUP> <FIELD LIST> <SLASH GROUP> ) 
1 173  <FIELD LIST> ::= <FIELD> 
2 174 / <FIELD LIST> <SEP> <FIELD> 
10 175  <SLASH GROUP> ::= / 
10 176 / <SLASH GROUP> / 
1 177  <FIELD> ::= <BASIC FIELD> 
1 178 / <GROUP> 
10 179  <SEP> ::= ,
10 180  / <SLASH GROUP>
1 181  <BASIC FIELD> ::= <HCONST>  
1 182  / <FMTFLD>
32 183  <GROUP> ::= <DCONST> <FMT>
1 184  / <FMT>
76 185  <FMTFLD> ::= FMTFLD
33 186  <FOA> ::= <ARRAY REF> = <EXPRESSION>
23 187  <ARRAY REF> ::= <NAME> ( <EXPRN LIST> )
1 188  <EXPRESSION> ::= <L TERM>
63 189  / <EXPRESSION> OR <L TERM>
88 190  <EXEC> ::= DO <LABEL> <NAME> = <DO PARAMETERS>
34 191  / IF ( <EXPRESSION> ) <BASIC>
34 192  / IF ( <EXPRESSION> ) <ARR ASMT>
34 193  / IF ( <EXPRESSION> ) <LOG IF LABELS>
41 194  / IF ( <EXPRESSION> ) <ARITH IF LABELS>
35 195  / ENTRY <NAME>
1 196  / <BASIC>
12 197  <DO PARAMETERS> ::= <INTEGER> , <INTEGER>
13 198  / <INTEGER> , <INTEGER> , <INTEGER>
37 199  <BASIC> ::= <NAME> = <EXPRESSION>
38 200  / ASSIGN <LABEL> TO <NAME>
39 201  / GOTO <LABEL>
40 202  / GOTO ( <LABEL LIST> ) , <EXPRESSION>
90 203  / GOTO <NAME> , ( <LABEL LIST> )
90 204  / GOTO <NAME> ( <LABEL LIST> )
43 205  / CALL <NAME> ( <EXPRN LIST> )
42 206  / CALL <NAME>
44 207  / RETURN
45 208  / CONTINUE
46 209  / STOP
47 210  / STOP <OCONST>
48 211  / PAUSE
49 212  / PAUSE <OCONST>
50 213  / REWIND <INTEGER>
51 214  / BACKSPACE <INTEGER>
52 215  / ENDFILE <INTEGER>
53 216  / READ <FORMATTED IOSPEC>
53 217  / READ <UNFORMATTED IOSPEC>
54 218  / WRITE <FORMATTED IOSPEC>
54 219  / WRITE <UNFORMATTED IOSPEC>
54 220  / PRINT <FORMATTED IOSPEC>
54 221  / PUNCH <FORMATTED IOSPEC>
55 222  / ENCODE <CODE SPEC> <IO LIST>
56 223  / DECODE <CODE SPEC> <IO LIST>
37 225  <ARR ASMT> ::= <ARRAY REF> = <EXPRESSION>
12 225  <LOG IF LABELS> ::= <LABEL> , <LABEL>
13 226  <ARITH IF LABELS> ::= <LABEL> , <LABEL> , <LABEL>
11 227  <LABEL LIST> ::= <LABEL LIST*>
11 228  <EXPRN LIST> ::= <EXPRN LIST*>
72 229  <OCONST> ::= OCONST
2 230  <FORMATTED IOSPEC> ::= <XFORMATTED SPEC> <IO LIST>
1 231  / <FORMATTED SPEC>
1 232  / <FORMATTED XIOSPEC>
2 233  <UNFORMATTED IOSPEC> ::= <UNFORMATTED SPEC> <IO LIST>
1 234  / <UNFORMATTED SPEC>
60 235  <CODE SPEC> ::= ( <INTEGER> , <FORM> , <NAME> )
61 236  <IO LIST> ::= <IO LIST*>
1 237  <LABEL LIST*> ::= <LABEL>
2 238  / <LABEL LIST*> , <LABEL>
58 239  <XFORMATTED SPEC> ::= ( <INTEGER> , * )
1 240  / <FORMATTED SPEC>
59 241  <FORMATTED SPEC> ::= ( <INTEGER> , <FORM> )
2 242  <FORMATTED XIOSPEC> ::= <XFORMATTED XSPEC> , <IO LIST>
1 243  / <FORMATTED XSPEC>
57 244  <XFORMATTED XSPEC> ::= *
58 245  / <FORM>
58 246  <FORMATTED XSPEC> ::= <FORM>
58 247  <UNFORMATTED SPEC> ::= ( <INTEGER> )
1 248  <FORM> ::= <LABEL>
1 249  / <NAME>
1 250  <IO LIST*> ::= <NAMED VALUE>
1 251  / ( <NAMED VALUE LIST> )
1 252  / ( <ITERATION LIST> )
2 253  / <NAMED VALUE> , <IO LIST*>
2 254  / ( <NAMED VALUE LIST> ) , <IO LIST*>
2 255  / ( <ITERATION LIST> ) , <IO LIST*>
1 256  <NAMED VALUE> ::= <NAME>
1 257  / <ARRAY REF>
<ITERATION LIST> ::= <ITERATION LIST*>

<ITERATION LIST*> ::= <NAMED VALUE> , <DO SPECIFICATION>
/ ( <NAMED VALUE LIST> ) , <DO SPECIFICATION>
/ ( <ITERATION LIST> ) , <DO SPECIFICATION>
/ <NAMED VALUE> , <ITERATION LIST*>
/ ( <NAMED VALUE LIST> ) , <ITERATION LIST*>
/ ( <ITERATION LIST> ) , <ITERATION LIST*>

<DO SPECIFICATION> ::= <NAME> = <DO PARAMETERS>

<L TERM> ::= <L FACTOR>
/ <L TERM> AND <L FACTOR>

<L FACTOR> ::= <L PRIMARY>
/ NOT <L PRIMARY>

<L PRIMARY> ::= <LCONST>
/ <RELATIONAL EXPRN>
/ <ARITH EXPRN>

<RELATIONAL EXPRN> ::= <ARITH EXPRN> <RELOP> <ARITH EXPRN>

<ARITH EXPRN> ::= <A TERM>
/ + <A TERM>
/ - <A TERM>
/ <ARITH EXPRN> + <A TERM>
/ <ARITH EXPRN> - <A TERM>

<RELOP> ::= RELOP

<A TERM> ::= <A FACTOR>
/ <A TERM> * <A FACTOR>
/ <A TERM> / <A FACTOR>

<A FACTOR> ::= <A PRIMARY>
/ <A PRIMARY> ** <A PRIMARY>

<A PRIMARY> ::= <NUMBER>
/ <NAMED VALUE>
/ <COMPLEX CONST>
/ <HCONST>
/ ( <EXPRESSION> )

<ICONST> ::= DCONST
/ OCONST

<RCONST> ::= RCONST

<DPCONST> ::= DPCONST

<CCONST ELEMENT> ::= <RCONST>
/ + <RCONST>
/ - <RCONST>
1 297  `<EXPRN_LIST*> ::= <EXPRESSION>
2 298  / <EXPRN_LIST*>, <EXPRESSION>
THE GRAMMAR IS LALR1
**special error**

**erroorno :** 0

**erroorno :**

**expected symbol:**

<table>
<thead>
<tr>
<th>Erron</th>
<th>DConst</th>
<th>BlockData</th>
<th>Program</th>
<th>Subroutine</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Implicit</td>
<td>Integer</td>
<td>REAL</td>
<td>DoublePrec</td>
<td>Complex</td>
</tr>
<tr>
<td>2</td>
<td>Logical</td>
<td>Dimension</td>
<td>COMMON</td>
<td>EQUIVALENCE</td>
<td>ExTERNAL</td>
</tr>
<tr>
<td>3</td>
<td>Level</td>
<td>DATA</td>
<td>DO</td>
<td>IF</td>
<td>ENTRY</td>
</tr>
<tr>
<td>4</td>
<td>Assign</td>
<td>Goto</td>
<td>CALL</td>
<td>RETURN</td>
<td>Continue</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
<td>PAUSE</td>
<td>Rewind</td>
<td>BACKSPACE</td>
<td>EndFile</td>
</tr>
<tr>
<td>6</td>
<td>Read</td>
<td>Write</td>
<td>Print</td>
<td>Punch</td>
<td>ENCODE</td>
</tr>
</tbody>
</table>

2 : (  
3 : DConst OConst NAME  
4 : ,  
5 : DConst NAME  
6 : NAME  
7 : )  
8 : ( NAME  
9 : NOT HConst LConst + - (  
   DConst OConst RConst DPCost  
10 : ( DPCost NAME  
   HConst DConst OConst RConst  
11 : + - ( HConst DConst  
   OConst RConst DPCost NAME  
12 : LConst + - ( HConst  
   DConst OConst RConst DPCost NAME  
13 : + - RConst  
14 : RConst  
15 : ) OR  
16 : ( * DConst NAME  
17 : * DConst NAME  

18 : ,
19 : ( DCONST NAME
20 : ,
21 : DCONST
22 : TO
23 : =
24 : ASSIGN GOTO CALL RETURN CONTINUE
   STOP PAUSE REWIND BACKSPACE ENDFILE
   READ WRITE PRINT PUNCH ENCODE
   DECODE NAME DCONST
25 : =
26 : EOS
27 : /
28 : DCONST HCONST LCONST + -
   ( OCONST RCONST DPCONST
29 : DCONST OCONST RCONST DPCONST
30 : *
31 : /
32 : NAME
33 : DO IF ENTRY DATA
   ASSIGN GOTO CALL RETURN CONTINUE
   STOP PAUSE REWIND BACKSPACE ENDFILE
   READ WRITE PRINT PUNCH ENCODE
   DECODE NAME
34 : ) /
35 : ) /
36 : HCONST FMTFLD DCONST ( INTEGER REAL DOUBLEPREC COMPLEX LOGICAL
37 : ) -
38 : FUNCTION NAME
39 : DCONST DATA DO IF ENTRY
   ASSIGN GOTO CALL RETURN CONTINUE
<table>
<thead>
<tr>
<th>STOP</th>
<th>PAUSE</th>
<th>REWIND</th>
<th>BACKSPACE</th>
<th>ENDFILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>WRITE</td>
<td>PRINT</td>
<td>PUNCH</td>
<td>ENCODE</td>
</tr>
<tr>
<td>DECODE</td>
<td>NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

40:
EOS
(

41:
DCONST  | DIMENSION  | COMMON  | EQUIVALENC | EXTERNAL  |
_LEVEL  | DATA       | DO      | IF        | ENTRY    |
_INTEGER| REAL       | DOUBLEPREC | COMPLEX | LOGICAL |
_ASSIGN | GOTO       | CALL    | RETURN    | CONTINUE |
_STOP   | PAUSE      | REWIND  | BACKSPACE | ENDFILE |
_READ  | WRITE      | PRINT   | PUNCH     | ENCODE |
| DECODE| INTEGER    | REAL    | DOUBLEPREC | COMPLEX |
| LOGICAL| NAME      |         |           |         |

42:
FORMAT  | DIMENSION  | COMMON  | EQUIVALENC | EXTERNAL  |
_LEVEL  | DATA       | DO      | IF        | ENTRY    |
_ASSIGN | GOTO       | CALL    | RETURN    | CONTINUE |
_STOP   | PAUSE      | REWIND  | BACKSPACE | ENDFILE |
_READ  | WRITE      | PRINT   | PUNCH     | ENCODE |
| DECODE| INTEGER    | REAL    | DOUBLEPREC | COMPLEX |
| LOGICAL| NAME      |         |           |         |

43:
DCONST  | IMPLICIT | DIMENSION  | COMMON  | EQUIVALENC |
_EXTERNAL| LEVEL    | DATA      | DO      | IF        |
_ENTRY  | INTEGER  | DOUBLEPREC | COMPLEX |
_LOGICAL| GOTO     | CALL      | RETURN  |         |
_CONTINUE| STOP     | PAUSE     | REWIND  | BACKSPACE |
_ENDFILE| READ     | WRITE     | PRINT   | PUNCH    |
| ENCODE| DECODE   | INTEGER   | REAL    | DOUBLEPREC |
| COMPLEX| LOGICAL  | NAME     |         |         |

44:
FORMAT  | IMPLICIT | DIMENSION  | COMMON  | EQUIVALENC |
_EXTERNAL| LEVEL    | DATA      | DO      | IF        |
_ENTRY  | ASSIGN   | GOTO      | CALL    | RETURN    |
_CONTINUE| STOP     | PAUSE     | REWIND  | BACKSPACE |
_ENDFILE| READ     | WRITE     | PRINT   | PUNCH    |
| ENCODE| DECODE   | INTEGER   | REAL    | DOUBLEPREC |
| COMPLEX| LOGICAL  | NAME     |         |         |

45:
END

46:
/   DCONST  NAME

47:
EOS   NAME

48:
END  DCONST  DIMENSION  COMMON  EQUIVALENC
_EXTERNAL| LEVEL | DATA | INTEGER | REAL |
_DOUBLEPREC| COMPLEX | LOGICAL |

49:
DIMENSION  COMMON  EQUIVALENC  EXTERNAL  LEVEL
_DATA  INTEGER | REAL | DOUBLEPREC | COMPLEX |
_LOGICAL |

50:
END  DCONST  DATA
51:    DATA

52:    FORMAT  BLOCKDATA  IMPLICIT  DIMENSION  COMMON
        EQUIVALENCE  EXTERNAL  LEVEL  DATA  DO
        IF  ENTRY  SUBROUTINE  ASSIGN  GOTO
        CALL  RETURN  CONTINUE  STOP  PAUSE
        REWIND  BACKSPACE  ENDFILE  READ  WRITE
        PRINT  PUNCH  ENCODE  DECODE  FUNCTION
        INTEGER  REAL  DOUBLEPREC  COMPLEX  LOGICAL
        NAME

53:    -EOF-
        DCONST  BLOCKDATA  PROGRAM  SUBROUTINE
        FUNCTION  IMPLICIT  INTEGER  REAL  DOUBLEPREC
        COMPLEX  LOGICAL  DIMENSION  COMMON  EQUIVALENCE
        EXTERNAL  LEVEL  DATA  DO  IF
        ENTRY  ASSIGN  GOTO  CALL  RETURN
        CONTINUE  STOP  PAUSE  REWIND  BACKSPACE
        ENDFILE  READ  WRITE  PRINT  PUNCH
        ENCODE  DECODE  NAME

******************************************************************************
## Storage Requirements

<table>
<thead>
<tr>
<th>Const</th>
<th>Allocated</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>350</td>
<td>299</td>
</tr>
<tr>
<td>2</td>
<td>240</td>
<td>197</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>5000</td>
<td>3380</td>
</tr>
<tr>
<td>6</td>
<td>7000</td>
<td>996</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4689</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>800</td>
<td>133</td>
</tr>
<tr>
<td>10</td>
<td>850</td>
<td>720</td>
</tr>
<tr>
<td>11</td>
<td>160</td>
<td>134</td>
</tr>
<tr>
<td>12</td>
<td>300</td>
<td>228</td>
</tr>
<tr>
<td>14</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>800</td>
<td>475</td>
</tr>
<tr>
<td>22</td>
<td>500</td>
<td>390</td>
</tr>
</tbody>
</table>

**Time Used:** 81638 MSEC
References


Appendix A: Modifications to BOBSW

To produce CLEMSW, the following modifications (by section of BOBSW manual) were made to BOBSW.

1. The system generates tables in the form of a FORTRAN block data subprogram named PARSBD which is compiled with a FORTRAN program. The FORTRAN program consists of a lexical scanner and a parsing algorithm with error recovery.

3. Only the sections LEXEMES and PRODUCTIONS have been kept in CLEMSW.
3.5 The lexemes must be in the order specified in the lexical tables (block data subprogram, TKTPBD) produced by FSCAN. The first lexeme, EOFTOK, is automatically included by CLEMSW and should not be specified in the lexeme list.

4. The lexical scanner is produced by the FSCAN system.

6. The parser program is written in FORTRAN. It employs a scanner produced by the FSCAN system.

Appendix A. Options 1 and 22 are permanently set, i.e., the grammar is treated as LALR(1) and simple productions are never removed.

Supplement 1. Option 27 is permanently set, i.e., semantic labels are user specified.

Modifications to the code not dealt with in the manual.
- CONSTREL is deleted in CLEMSW
- The set, LEX, in the procedure ADEQUAD is converted to an array of logicals to allow greater than 59 lexemes.
- The programming error in the procedure OUTGRAMMAR is fixed in CLEMSW (the error occurs when the PASCAL compiler produces code to check array bounds on reference).