```
grammar
   | cprog> - <modDecs> <othermods> <driver> <othermods>
  2 <modDecs> - <modDec> <modDecs>
3 <modDecs> - eps
4 <modDec> - DECLARE MODULE ID SEMICOL
   5<othermods> - <mod> <othermods> 6<othermods> - eps
   <driver> - DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <modDef>
   g <mod> - DEF MODULE ID ENDDEF TAKES INPUT SQBO <ip_list> SQBC SEMICOL <ret>
   @ <modDef>
   <ret> - RETURNS SQBO <op_list> SQBC SEMICOL
cret> - eps
   12<ip_list> - ID COLON <dataType> <N1>
  N1> - COMMA ID COLON <dataType> <N1> N1> - eps Op_list> - ID COLON <type> <N2>
   SKN2> - COMMA ID COLON <type> <N2>
   N2> - eps
  dataType> - INTEGER & dataType> - REAL
 gq<dataType> - BOOLEAN
 20 <dataType> - ARRAY SQBO <range_arrays> SQBC OF <type>
 21 < range_arrays> - < index_arr> RANGEOP < index_arr>
 22 < type> - INTEGER
 23<type> - REAL
 54<type> - BOOLEAN
 5<modDef> - START <stmts> END
    <stmts> - <stmt> <stmts>
24 <stmts> - eps
 28 <stmt> - <iostmt>
 20<stmt> - <simplestmt>
30<stmt> - <declarestmt>
  si<stmt> - <condstmt>
 32<stmt> - <iterstmt>
 33<iostmt> - GET_VALUE BO ID BC SEMICOL
 34<iostmt> - PRINT BO <var_print> BC SEMICOL
 35 < boolconst> - TRUE
 36<boolconst> - FALSE
 37<id_num_rnum> - ID
 38<id_num_rnum> - NUM
34<id_num_rnum> - RNUM
40<arr_ele_print> - ID SQBO <index_arr> SQBC
41 <var_print> - ID <P1>
 42 < var_print> - NUM
 43
43
44
var_print> - <boolconst>
45<P1> - SQBO <new_index> SQBC
46<P1> - eps
47<simplestmt> - <assignstmt>
५६<simplestmt> - <modReusestmt>
44<assignstmt> - ID <whichstmt> 50<whichstmt> - <lvalueIDstmt> 51 <whichstmt> - <lvalueARRstmt>
52 < |valueIDstmt> - ASSIGNOP < exp> SEMICOL
c,3 < lvalueARRstmt> - SQBO <element_index_with_expressions> SQBC ASSIGNOP <exp>
   SEMICOL
54 <index_arr> - <sign> <new_index>
 55 < new_index> - NUM
 56 < new_index> - ID
 57<sign> - PLUS
 58<sign> - MINUS
 5q <sign> - eps
 60 < modReusestmt> - < optional> USE MODULE ID WITH PARAMETERS < paralist> SEMICOL
 6 ( <paralist> - <sign> <actual_para_list> <paralist2>
 62 <paralist2> - COMMA <sign> <actual_para_list> <paralist2>
63 <paralist2> - eps
64 <actual_para_list> - NUM
65 <actual_para_list> - RNUM
    <actual_para_list> - <boolconst>
```

```
grammar
         67 <actual_para_list> - ID <N_11>
         cs<optional> - SQBO <idList> SQBC ASSIGNOP
       64<optional> - eps
           70<idList> - ID <N3>
         71<N3> - COMMA ID <N3>
72<N3> - eps
      13 <exp> - <arithorboolExp>
    2 4 <exp> - <U>
       75<U> - <unary_op> <new_NT>
       76 < unary_op> - PLUS
77 < unary_op> - MINUS
       78 < new_NT> - BO < arithExp> BC 
70 < new_NT> - < id_num_rnum>
       80<arithOrboolExp> - <AnyTerm> <N4>
       8 < N4> - < logOp> < AnyTerm> < N4> 8 z < N4> - eps
     83 <AnyTerm> - <arithExp> <N5>
84 <AnyTerm> - <boolconst>
       85 <N5> - <relop> <arithExp>
      03 <arithExp> - <term> <N6> 08 <N6> - <op1> <term> <N6> 80 <N6> - <op1> <term> <N6> 80 <N6> - <op1> <term> <N6> 80 <N6> - <op1
         40 <term> - <factor> <N7>
        a1<N7> - <op2> <factor> <N7> 42<N7> - eps
       q3<factor> - BO <arithorboolExp> BC
       a4 <factor> - NUM
        q5<factor> - RNUM
q6<factor> - ID <N_11>
       q7 <N_11> - SQBO <element_index_with_expressions> SQBC
      control c
   <element_index_with_expressions> - <op1> <N_10>
    | <element_index_with_expressions> - <arrExpr>
    102 <N_10> - <new_index>
   103 <N_10> - BO <arrExpr> BC
(07<arrTerm> - <arrFactor> <arr_N5>
    108 <arr_N5> - <op2> <arr_N5> - <arr_N5> - eps
    110 <arrfactor> - ID
     | | | <arrFactor> - NUM
     1 < arrFactor> - < boolconst>
      13 <arr Factor> - BO <arr Expr> BC
       114 < Op1> - PLUS
       115<0p1> - MINUS
      116 < Op 2> - MUL
      17 <0p2> - DIV
         120<re1op> - LT
         121<relop> - LE
         132<relop> - GT
        123<relop> - GE
124<relop> - EQ
125<relop> - NE
          26<declarestmt> - DECLARE <idList> COLON <dataType> SEMICOL
          128 Condstmt> - SWITCH BO ID BC START <caseStmts> <default> END 128 CaseStmts> - CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 139 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> BREAK SEMICOL <N9> 130 < CASE <value> COLON <stmts> 130 < CASE <value> COLON <stmts> 130 < CASE <value> COLON <stmts> 130 <cmts> 130 <cmts>
         130 < N9> - eps
        131 <value> - NUM

<pre
       134<default> - DEFAULT COLON <stmts> BREAK SEMICOL
                                                                                                                                                                                    Page 2
```

grammar

|35</br>
|35</br>
|36</br>
|36</br>
|36</br>
|37</br>
|37</br>
|38</br>
|38</br>
|38</br>
|39</br>
|30</br>
|30 443<sign_forloop> - eps

```
SYNTAX TREE (AST) CREATION.
 MEMBERS
            ID; 201984A7
RAJAN
             05721
YASH
                               NOTE: - all rules transsal order: POST ORDER
             0638P
 AYUSH
                                                              traversal.
             0652P
                                   6- assume that *-syn attributes are first
VASU
             0656P
SUDARSHAN
             0744P
          - { (Semantic Rules > }, Format | created (in post-order fashion).
 1. { L POST ORDER > :
      make-node ("MODULEDECLARATONS", <mod Decs). list-syn),
                        make-node ("OTHERMODULES", CotherMods ). let-syn),
                        make-node ("DRIVER", < doriver). node-syn),
                        make-node ("OTHERMODULEST", Cother Mods > list-syn));
   free ((mod Decs)); free (Cother Mods)); free ((doiver)); free (Cother Mods), }
2. { L POST. ORDER > :
     <modDecs >. list-syn = insert-at-head (<modDecs >, list-syn, (modDec > node-syn);
   free (< mod Dec>); free (< mod Decs >1); }
3. { L POST-ORDER ) :
    (mod Decs ). list-syn = NULL
    free (eps); }
4. { C POST-ORDER > :
    (mod Dec). node-syn = 10. addr 3
    free (DECLARE); free (MODULE); free (SEMICOL);}
5. { CPOST-ORDER > :
    (other mods). list-syn = insert-at-head (Cother mods), list-syn,
                                           (mod). mode -syn) ;
   free (Lmod);
   free (cother Mods),); }
```

SEMANTIC RULES FOR ABSTRACT

GROUP-20

```
6. { C POST- ORDER > 1
       Cother mods >, list-syn = NULL;
       free (eps);
 7. { LPOST-ORDER > :
      (driver). node-syn = (mod Deb). node-syn
    free (DRIVERDEF); free (DRIVER); free (PROGRAM); free (DRIVER-ENDDEF);
   free ((modDeb>); }
8, { LPOST-ORDER):
     (mod). node-syn = make-node (ID. name, <ip-list). list-syn,
                                   (ret). list-syn, (mod Deb). node-syn);
   free (DEF); free (MODULE); free (ENDDEF); free (TAKES); free (INPUT);
  free ( SQBO); free ( Cip-plist)); free (SQBC); free (SEMICOL); free (Cret));
  free ( (mod Def ) ); }
9. { LPOST-ORDER):
    Cret >. list-syn = Cop-plist>. list-syn ;
  free (RETURNS); free (SQBO); free (SQBC); free (COP-Plist);
  free (SEMICOL); }
10. } LPOST- OR DER):
    <ret>. list_syn = NULL
    free (eps);}
11. { CPOST-ORDER):
   Cip-pust>, node-syn = make-node (ID. name, Edala Type) = node-syn);
   Cip-plist>. list-syn = insert-at-head (<N1>.list-syn, <ip-plist>.node-syn);
   free (LOLON); free (Ldata Tyre); free (<N1>); }.
12. { CPOST-ORDER > :
  (N1). node-syn = make-node (ID. name, Edate Tyre). node-syn);
  <N1>. list-syn = insert_at_head (<N1>,-list-syn, <N1>. node-syn);
  free(comma); free (colon); free (cdotaTyn); free (<N1>1); }
```

```
13. { <N1>. lict-syn = NULL; free (CPS); }
 14. { <op-list>. node-syn = make-node (10. name, Ctyre). node-syn);
       Cop-list >. list-syn = insert-at-head (<N2>.1ist-syn, Cop-list >. node-syn);
       free (COLON); free (Krypis); free (KN2>); }
 15. { (N2), node-syn = make-node ( ID. name, (type), node-syn) >
        <N2). list-syn = insert-at-head ( <N27, list-syn, <N2), node-syn);
        free (comma) ; free (colon); free (Kryne>); free (KN2),) }
 16. { <NZ>. list-syn = NULL 3
        free (eps); ].
     { Ldota Type > . node - syn = INTEGER. addr ; }
      { (data Type > , node - syn = REAL , addr ; )
     { (do to Type), node-syn = BOOLEAN. oddr ?}
20.
     { (data Type > node -syn= make-node ("ARRAY", Crange-orrays > node-syn)
                                                (tyne>, node-syn);
     free (Crange-arrays);
      free ( <type >); free (ARLAY); free (SQCO); free (SQBC); free (OF); }
21. { (range-arrays), node-syn = makenode ("ARRAY RANGE", (index-array), node
                                             , Lindex-away >2. node-syn);
      free (Kinder-array >1);
     free (Cindex-array) 2);
     Free (RANGEOP); }
    { < type > node-syn = INTEGER. addr ; }
     { < type > node - syn = REAL , addr ; }
     { < rype > , node-syn = BOOLEAN , addr ; }
25. { Emod Def 2. node-syn = make node ("STATEMENTS", < stm+>. list - syn);
     free (START); free (<stmt+)); free (END); }
26. { < stmts >, list-syn = insert-at-head (< stmts), list-syn, < stmt>. node-syn);
    free (<stm+>); free (cstm+s>); }
```

```
27. { < stmts >. list-syn = NULL; free (eps) ;}
28. { < stm+> node-syn = <iochm+> node-syn >
      free (ciostm+>); }
29. { (stant). node-syn = (simplestant), node-syn; free ((simplestant));}
30. { < stm+>. node-syn = {declarestm+>. node-syn; free ( {declarestm+>); }
31. { (stm+), node-syn = (cond stm+), node-syn; free ((condstm+)); }
32. { (stm+). node-syn = (iterstm+), node-syn; free ((iterstm+)); }
33. { Liostnt >. node-syn = makemode ("INPUT", ID. addr);
      free (GET_VALUE); free (BO); free (BC); free (SEMICOL) }
34. { Liostmt >. node-syn = (var-print >. node -syn >
     fru ( PRINT); free (BO); free (BC); free (SEMICOL); }
35. { (boolconst ), node-syn = TRUE, addr is ]
36. { (boolconst). node-syn = FALSE-addr; }
37. { (id-num-rnum), node-syn = ID. addris
38. { (id-num_rnum). mode-syn = NUM. addr; }
39. { <id_num_ rnum > . node-syn = RNUM . addr; }
40. / REDUNDANT RULE
41. { < P1). node-inh = 7D. addr ;
     Ever-print >. node-syn = (P1>, node-syn; free (<P1>); }
42, { Evar-point >, node-syn = NUM. addr ; }
43. { (var-print) = node -syn = RNUM addi; }
44. { Evar-print >. mode-syn = eboolconst >. mode-syn; free (Eboolconst >); }
45. { <P1>, node-syn = make-node ("ARRAY_", <P1>, node-inh >
                                              (new-index), node -syn);
    free (SQBO); free ((new-index));
    free (sabc);}
46. { < P1>, node-syn = < P1>, node-inh ; free (ops) ; }
47. { (simplestmt >, node-syn : (assignstmt >. node-syn ; free ((assignstmt)); }
48. { < simplestmt > . node - syn = < modReusestmt > . node - syn ; free ( 2 mod Reuse strut > ); }
49. { < which stmt > node - inh = ID . addr;
      Cassignstmt). node-syn = (whichstmt), node-syn
      free ((which start >); }
```

```
50. { < Ivalue ID start > . node - inh = < which start > . node - inh ;
     < which stm+ > . node-syn = < Ivalue IDstm+ > . node-syn >
     free ( ( lvolve ID start >); }
51. { < I value ARRstm+>, node-inh = ( which stm+ ). node-inh;
      (whichsom+) - mode-syn = (Ivalue ARR stont) - mode-syn;
     free ( < ( value ARRston + > ); }
52. { < Ivalue IDstrot >. node-syn = make-node ("Assign", < Ivalue IDstrot >. node-inh,
                                                          (exp). node-syn);
     free ((exps);
     free (ASSIGNOP);
     free (SEMICOL); }
53. { < | value ARRstmt > node-syn = make-node ("ASSIGN", Lelement-index-with-expr).
                                          , (expsinade-syn);
   free ((exps);
  free (SQBO); free (SQBC); free (ASSIGNOP); free (SEMICOL);
   free ((element-index-wth-expr)); free ((exp)); }
54.
55. { (new-index). node-syn = NUM. addr; }
56. { (now-index). node-syn = ID. addr; }.
57. { <sign >. node - syn = PLUS. addr; }
58. { < sign > . mode-syn = MINUS addr; }
59. { < cign > node-syn = NULL; }
60. { (mod Reuse stm+). node-syn = makenode ("Modulereuse", 1D. adde,
                                              Coptional >. list-syn, Sparalist> list-syn);
   free ((optional));
   free (USE); free (MODULE); free (WITH);
   free (PARAMETERS); free (LPONALIST); free (SEMICOL); f
```

```
61. { Eporolist > node-syn = make-node ("PARAMETER", <sign> node-syn ,
                                     <octual-para-list > node-syn) >
     cparalist>. list-syn = insert-at-head (Cparalist>>, list-syn , cparalist>. node-syn
    free ((sign)); free (< actual-para-list >); free (<paralist 2)); }
 62. { (paralist 2). node-syn = make-node ("PARAMETER", (sign). node-syn,
                                       (actual-para-list). node-syn);
    free (comma); free ((sign)); free ((actual-para-list)); free ((parallet 2));
63. { (pasalist 2). list-Syn = NULL; free (eps);}
64. { (octual-para-list). node-syn = NUM. addr ; }
65. { Loctual-para-list >. node-syn = RNUM. addr; }
66. { (actual-para-list), mode-syn = (boolconst), mode-syn ; free ((boolconst));}
67. { (actual-para-list). node-syn = makenode ("PARAM-", ID. addr,
                                              <N11), node-syn);
    free (< N11>); }
68. {Cortional > . list-syn = <id List > . list-syn; free (SQBO); free (SQBC);
    free (ASSIGNOP); free (CidLists); }.
69. { Coptional >. list-syn = NULL;
    free (eps); ]
70. { Lidust > node-syn = ID. addr;
      CidList>. list-syn = insert-at-head (<N3>, list-syn, <idList>. mode-syn)
      free ((N3)); }
71. { < N3>. node-syn = ID. addr;
       <N3). List-syn = insort_at-head (<N3), list-syn, <N3), node-syn);</pre>
      free ((N3)); free (comma); 3
72. { (N3>. list-syn = NULL )
      free (eps); }.
73. { lexp>, node-syn = (arithorboolExp), node-syn;
     free ( Carithor bool Exp) ) > ]
 74. { Lexpr. node-syn = (U), node-syn; free ((U));}
```

```
75. { (new-NT), node-inh = (unary-op), node-syn ;
      (U) node-syn = (now-NT), node-syn;
     True ( ( ( nary-op)); free ( (new-NT)); }
 76. { <unary-op>. node-syn = PLUS. addr; }
 77. { cunary-op >. node-syn = MINUS addr; }
 78. { <new-NT>. node-syn = makenode ("UNARYASSIGN", <new-NT). node-inh,
                                             Conthexps. node-syn);
     free (BO);
     free ((arith Exp));
     free (BC);}
 79. { (new-NT), node-syn = makemode ("UNARY ASSIGN", (new-NT), node-inh,
                                           (id-num-rnum ). node-syn);
    free ( <id-num-rnum >);}
80. { <N4>, node-inh = <AnyTerm>, node-syn; 

<anithOrboolExp>, node-syn = <N4>, node-syn;
      free (< Any Term?); free (<N4>);}
81. { < N4>, node inh = makenode (logop) node syn < N4), node inh, < Any Tem), node -syn
       <N4>, node-syn = <N4>, node-syn
      free ((N4));
      free (<10g0P>);
free (<Ary Term>); }
82. { < Nu>, node-syn = < N4>, node-inh; }
83. { LNS7. mode-inh = Lorith Exp), node-syn;
      (Any Term). node - syn = (NS). node - syn ;
      free (conth Exp>);
     free ((N5)); 3
84. { (Any Term), node-syn = (bookenst), node-syn; free ((bookenst)); }
85. { < N5>. node-syn = makenode ( < relop). node-syn, < N5>, node-inh,
                                     (arrithExp), node-syn) j
      free ((relop));
      free ( (arith Emp)); }
```

```
86. ( (NS), node-syn = (NS), node-inh; free (eps); }
 87. { <N6>, node-inh = <+erm>, node-syn;
      CorithExp), node-syn = <N6), node-syn ;
       free ( < term > );
      free (< N6>); }
88. { (N6), node-inh = make-node ( (op1), node-syn, (N6), node-inh, (kem), nodesy)
      <NG>. node-syn = <NG>, node-syn;
      free ((op1)); free ((N6)); free ((+em)); }
89. { < N6>, node-syn = < N6>, node-inh;
      free (eps); ]
90. { (N7), node-inh = (foctor), node-syn;
       (term) - node - syn = (N7), node - syn ;
       free ((foctor));
      free ( ( N7)); }
31. { < N7}, node-inh = make_node ( <op2 >. node-syn, <N7>, node-inh,
                                  (foctor) node-syn);
     <N7). node-syn = <N7), node-syn;
    free ( < 0p2>);
    free ( (factor));
    free ( < N7 > 1); }
92, { (N7), node_syn = (N7), node-inh; free (eps); }
93. { (foctor), node-syn = (a with Or bool Exp), node-syn ?
     free ( 80); free (BC);
     free ((authorboolExp)); }.
94. { (factor > node-syn = NUM. addr ; )
95. { (factor). node-syn = RNUM. addr ;}
96. { (N-11). node-inh = ID. addr;
      (factor). node-syn = (N-11). node-syn >
       free ((N-11)); }
97. { (N-11). node-syn = make-node ('ARRAY-ACC', < NA1) node-inh,
                                      (element index with rexpo) node - Syn);
   free (SQBC); free (Celement-index-with-expr)); free (SQBO);}
```

```
98. { < N-11), node-syn= < N-11) - node-inh 's ]
 99. 1 REDUNDANT RULE .
100. { (N-10), node-inh = (op1), node-syn',
      free (COPAS); free (CN-10);
101. { Leternant - index - with expr ) , node - syn = Karr Expr > , node - syn ;
       free ((wrexpos));}
102. { < N-10>, node-syn = make-node ("UNI-NUMBER", < NLOD, node-inh,
                                     (now-index), node-syn);
      fra ((nav-index));}
103. { < N-10). node-syn = make-node ("UNI-ARR-NUM", < N-10). node-inh, < acrexpr). node-syn);
     free ((arr Expr));
    free (BO);
    free (BC);
104. { (arr=N4), node-inh = (orrTem), node-syn ;
      Carr Expr >. node-syn = (arr-N4), node-syn;
     free ( (arr Term >); free ((arr-N4)); }
105. { (arr-N4), node-inh = make-node (Cop1), node-syp, Corr-N4), node-inh,
                                                   (oun Tem), node-Syn);
      Carr-Nut. nose-syn = Carr. N47, node-syn.
      free ( (am Tern >); free ( (ann-N4), );}
106. { (arr-N4), node-syn = (arr-N4), node-inh 3}
107. { Lava_N5>. node-inh = Carafactor>. node-syn 5
      (arr Term) node-syn = (arr-N5) node-syn j
     free ( < on Factor ));
      free ( ( ( ( N5 )); }
108. { (au N5), mode-inh = make-node (Lopz), mode-syn, (ana N5), node-inh)
                                      (wor Factor), node - Syn);
     (ON - NE) . node-syn = (arm-N5), node-syn
     free ((ann-N5)); free ((ann-Factor)); }
```

```
109. { ( Dur- N5 ). node-syn = ( Dur- N5 ). node-inh i }.
 110. { Can Factor >, node-syn = ID. addr; }
 111. { (aux Factor), node-Syn = NUM, addrig?
 112. { (avor Factor). node-Syn = (boolconst), node-syn; free ((boolconst));}
113. { (an Factor), node-syn = (asr Expr), node-syn ; free (BO); free (BC);
      fice ((arr Expr)); }
     { Lop1). node-syn = PLUS. addr;}
      { <op 1> node - syn = MINUS. addr;}
      { (op2). node-syn = MUL.addr; }
116.
      { <op2 ). mode-syn = DIV.addr;}
117 .
     { < log Op> . node - Syn = AND . oddi; }
118.
      { < log Op>, node-syn = OR, addr; }
119.
      { (relop). mode-syn = LT. addr; }
120.
121.
      ( relop). node-syn = LE. addrij
22.
                             GT. add 733
     { (orlop), node-syn =
23.
     { < sclop > node - syn =
                             GE. addrig
     ( relop) node-syn =
                             EQ. addris
24.
     { (relop) node-syn = NE. addr; 3
    { Chedorestmt > . node-syn = make-node ("DECLARE", Cidlist> . List-syn ,
                                           LdataType>. nade-syn) j
   free (DECLARE); free (COLON); free (SEMICOL);
    free (CidList>); free (cdataType>); }.
127. { Coordsmt > , node-syn = make-node ("SWITCH", ID. addr,
                              (case storts ). list-syn, (default). node-syn);
   free (SWMCH); free (BC); free (Lone starts); free (END); free (BO);
    free (START); free (DEFAULT); }
128. { (casestats). node-syn = make-node ("CASE", (value). node - syn,
                                             (stmts). list_syn);
       (casestmts). List-syn = insert-of-head (<N9). Ust-Syn, (case stmts), node-syn);
      free (CASE); free (<volue>); free (colon); free (<stmt3>);
      free (BREAK); free (SEMICOL); free (<N9>);
```

```
129. { <Ng>, node-syn = make-node ("CASE", (volve), node-syn, (stmts), list-syn); 
<Ng>, list-syn = insert-at-head (<Ng>, (ist-syn , (core stmts), node-syn);
    Free (CASE); free (Evolus); free (COLON); free (CSAMES); free (BREAK)
    free (SEMICOL); free (CN9>1); ]
     { <N3>. list-syn = NULL; free (eps);}
131. { (volue), node - syn = NUM, addr 33
132. { (value), mode-syn = TRUE, add= ; }
133. { (volue). node-syn = FALSE-addr; }
134. { Ldufaut). node-syn = make-node ("DEFAULT CASE", (stmts). let-syn);
    Free (DEFAULT); fran (BREAK); free (CCHMIS)); free (COLON);
    free (SENICOL); }
135. { (default), node-syn = NULL; free (eps);}
136. { Literstrat > . node-syn = make-node ("FOR", ID. oddr,
                          (sage-for-loop). node-syn, (struts). list-syn)
  Sree (FOR); free (BO); free (ZN); free (< rage -for-loop)); fee (BC);
 free (START); free (END); free (Estmis);
free (WHILE); free (BO); free (CONHOLDODIEMP)); free (BC);
    free (START); free (CSMBS); free (END); }
38. { (rage-forloop ) node-syn =
                                   me lu-node ("FOR RANGE",
                                   <index-for loop). node-syng
                                   (inlex-fooloop) node- Syn)
    free ( cindex.for 100p));
    free ( Lindex-for loop) 2);
    free (RANGEOP); } .
139. { Cindex-forloop). node-syn =
                                  make-node (" FOR INDEX")
                                  (sign), node-syn, (new-index fooloop), node-syn)
     free (<sign-fooloop>); free (<rew-index-fooloop)); }
```

140, { < new-influx-fooloop>, node-syn = NUM.addr;}

141. { < sign-fooloop>, node-syn = PLUS.addr;}

142. { < sign-fooloop>, node-syn = MINUS.addr;}

143. { < sign-fooloop>, node-syn = MINUS.addr;}

143. { < sign-fooloop>, node-syn = NULL; free (eps);}.