# SEMANTIC RULES | GROUP_34

| | Grammar Production Rules | |
|---|---|---|
| Sl No. | Top Down | Bottom Up |
| 1 | <program> → <moduleDeclarations><otherModules(1)><driverModule><otherModules(2)> | |
| | | <program>.addr=createNewNode(label:program,<moduleDeclarations>.syn_addr,<otherModules(1)>.syn_addr,<driverModule>.addr,<otherModules(2)>.syn_addr) |
| | | free(<moduleDeclarations>) |
| | | free(<otherModules(1)>) |
| | | free(<driverModule>) |
| | | free(<otherModules(2)> |
| 2 | <moduleDeclarations> → <moduleDeclaration><moduleDeclarations> | |
| | | <moduleDeclarations>.addr=createNewNode(label:moduleDeclaration,<moduleDeclaration>.addr) |
| | | <moduleDeclarations>.syn_addr=addToStart(<moduleDeclaration>.addr,<moduleDeclarations>.syn_addr) |
| | | free(<moduleDeclaration>) |
| | | free(<moduleDeclarations>) |
| 3 | <moduleDeclarations> → ε | |
| | | <moduleDeclarations>.syn_addr=NULL |
| | | free(ε) |
| 4 | <moduleDeclaration> → DECLARE MODULE ID SEMICOL | |
| | | <moduleDeclaration>.addr=ID.addr |
| | | free(DECLARE) |
| | | free(MODULE) |
| | | free(SEMICOL) |
| 5 | <otherModules(1)> → <module><otherModules(2)> | |
| | | <otherModules(1)>.addr=createNewNode(label:moduleNode,<module>.addr) |
| | | <otherModules(1)>.syn_addr=addToStart(<otherModules(1)>.addr,<otherModules(2)>.syn_addr) |
| | | free(<module>) |
| | | free(<otherModules(2)> |
| 6 | <otherModules> → ε | |
| | | <otherModules>.syn_addr=NULL |
| 7 | <driverModule> → DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef> | |
| | | <driverModule>.addr=createNewNode(label:driverModule,<moduleDef>.addr) |
| | | free(DRIVERDEF) |
| | | free(DRIVER) |
| | | free(PROGRAM) |
| | | free(DRIVERENDDEF) |
| | | free(<moduleDef>) |
| 8 | <module> → DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret><moduleDef> | |
| | | <module>.addr=createNewNode(label:module,ID.addr,<input_plist>.syn_addr,<ret>.addr,<moduleDef>.addr) |
| | | free(DEF) |
| | | free(MODULE) |
| | | free(ENDDEF) |
| | | free(TAKES) |
| | | free(INPUT) |
| | | free(SQBO) |
| | | free(<input_plist>) |
| | | free(SQBC) |
| | | free(SEMICOL) |
| | | free(<ret>) |

| | | |
|---|---|---|
| | | free(<moduleDef) |
| 9 | <ret> → RETURNS SQBO <output_plist> SQBC SEMICOL | |
| | | <ret>.addr=<output_plist>.syn_addr |
| | | free(RETURNS) |
| | | free(SQBO) |
| | | free(<output_plist>) |
| | | free(SQBC) |
| | | free(SEMICOL) |
| 10 | <ret> → ε | |
| | | <ret>.addr=NULL |
| | | free(ε) |
| 11 | <input_plist> → ID COLON <dataType> <input_plist2> | |
| | | <input_plist>.addr=createNewNode(label:linkedListNode,ID.addr,<dataType>.addr) |
| | | <input_plist>.syn_addr=addToStart(<input_plist>.addr,<input_plist2>.syn_addr) |
| | | free(ID) |
| | | free(COLON) |
| | | free(<dataType>) |
| | | free(<input_plist2>) |
| 12 | <input_plist2(1)> → COMMA ID COLON <dataType> <input_plist2(2)> | |
| | | <input_plist2(1)>.addr=createNewNode(label:linkedListNode,ID.addr,<dataType>.addr) |
| | | <input_plist2(1)>.syn_addr=addToStart(<input_plist2(1)>.addr,<input_plist2(2)>.syn_addr) |
| | | free(ID) |
| | | free(COLON) |
| | | free(<dataType>) |
| | | free(<input_plist2>) |
| | | free(COMMA) |
| 13 | <input_plist2> → ε | |
| | | <input_plist2>.syn_addr=NULL |
| | | free(ε) |
| 14 | <output_plist> → ID COLON <type> <output_plist2> | |
| | | <output_plist>.addr=createNewNode(label:linkedListNode,ID.addr,<type>.addr) |
| | | <output_plist>.syn_addr=addToStart(<output_plist>.addr,<output_plist2>.syn_addr) |
| | | free(ID) |
| | | free(COLON) |
| | | free(<type>) |
| | | free(<output_plist2>) |
| 15 | <output_plist2(1)> → COMMA ID COLON <type> <output_plist2(2)> | |
| | | <output_plist2(1)>.addr=createNewNode(label:linkedListNode,ID.addr,<type>.addr) |
| | | <output_plist2(1)>.syn_addr=addToStart(<output_plist2(1)>.addr,<output_plist2(2)>.syn_addr) |
| | | free(ID) |
| | | free(COLON) |
| | | free(<type>) |
| | | free(<output_plist2>) |
| | | free(COMMA) |
| 16 | <output_plist2> → ε | |
| | | <output_plist2>.syn_addr=NULL |
| | | free(ε) |
| 17 | <dataType> → INTEGER | |
| | | <dataType>.addr = INTEGER |
| 18 | <dataType> → REAL | |

| | | |
|---|---|---|
| | | <dataType>.addr = REAL |
| 19 | <dataType> → BOOLEAN | |
| | | <dataType>.addr = BOOLEAN |
| 20 | <dataType> → ARRAY SQBO <arr_range> SQBC OF <type> | |
| | | <dataType>.addr=createNewNode(label:arrayDataType,<arr_range>.addr,<type>.addr) |
| | | free(ARRAY) |
| | | free(SQBO) |
| | | free(<arr_range>) |
| | | free(SQBC) |
| | | free(OF) |
| | | free(<type>) |
| 21 | <arr_range>→ <arr_index(1)>< RANGEOP <arr_index(2)> | |
| | | <arr_range>.addr=createNewNode(label:rangeOP,<arr_index(1)>.addr,<arr_index(2)>.addr) |
| | | free(RANGEOP) |
| | | free(<arr_index(1)>) |
| | | free(<arr_index(2)>) |
| 22 | <type> → INTEGER | |
| | | <type>.addr = INTEGER |
| 23 | <type> → REAL | |
| | | <type>.addr = REAL |
| 24 | <type> → BOOLEAN | |
| | | <type>.addr = BOOLEAN |
| 25 | <moduleDef> →START <statements> END | |
| | | <moduleDef>.addr=<statements>.syn_addr |
| | | free(START) |
| | | free(END) |
| | | free(<statements>) |
| 26 | <statements(1)>→ <statement> <statements(2)> | |
| | | <statements(1)>.syn_addr=addToStart(<statement>.addr,<statements(2)>.syn_addr) |
| | | free(<statement>) |
| | | free(<statements(2)>) |
| 27 | <statements>→ ε | |
| | | <statements>.syn_addr=NULL |
| 28 | <statement> → <ioStmt> | |
| | | <statement>.addr=<ioStmt>.addr |
| 29 | <statement> → <simpleStmt> | |
| | | <statement>.addr=<simpleStmt>.addr |
| 30 | <statement> → <declareStmt> | |
| | | <statement>.addr=<declareStmt>.addr |
| 31 | <statement> → <conditionalStmt> | |
| | | <statement>.addr=<conditionalStmt>.addr |
| 32 | <statement> → <iterativeStmt> | |
| | | <statement>.addr=<iterativeStmt>.addr |
| 33 | <ioStmt> → PRINT BO <print_var> BC SEMICOL | |
| | | <ioStmt>.addr=createNewNode(label:printOutput,<print_var>.syn_addr) |
| | | free(PRINT) |
| | | free(BO) |
| | | free(BC) |
| | | free(SEMICOL) |
| 34 | <ioStmt> → GET_VALUE BO ID BC SEMICOL | |

| | | |
|---|---|---|
| | | <ioStmt>.addr=createNewNode(label:getInput,ID.addr) |
| | | free(GET_VALUE) |
| | | free(BO) |
| | | free(BC) |
| | | free(SEMICOL) |
| 35 | <boolConst> →TRUE | |
| | | <boolConst>.addr=TRUE.addr |
| 36 | <boolConst> → FALSE | |
| | | <boolConst>.addr=FALSE.addr |
| 37 | <print_var> → ID <N1> | |
| | | <print_var>.addr = ID.addr |
| | <N1>.inh_addr=<print_var>.addr | |
| | | <print_var>.syn_addr=<N1>.syn_addr |
| | | free(<N1>) |
| 38 | <print_var> →  NUM | |
| | | <print_var>.syn_addr=NUM.addr |
| 39 | <print_var> → RNUM | |
| | | <print_var>.syn_addr=RNUM.addr |
| 40 | <print_var> →  <boolConst> | |
| | | <print_var>.syn_addr=boolConst.addr |
| 41 | <N1> → SQBO<sign><index2> SQBC | |
| | | <N1>.addr=createNewNode(label:signedNum,<sign>.addr,<index2>.addr) |
| | | <N1>.syn_addr=createNewNode(label:arrayElement,<N1>.inh_addr,<N1>.addr) |
| | | free(SQBO) |
| | | free(SQBC) |
| | | free(<sign>) |
| | | free(<index2>) |
| 42 | <N1> →  ε | |
| | | <N1>.syn_addr=<N1>.inh_addr |
| 43 | <simpleStmt>  →<assignmentStmt> | |
| | | <simpleStmt>.addr=<assignmentStmt>.syn_addr |
| | | free(<assignmentStmt>) |
| 44 | <simpleStmt>  → <moduleReuseStmt> | |
| | | <simpleStmt>.addr=<moduleReuseStmt>.syn_addr |
| | | free(<moduleReuseStmt>) |
| 45 | <assignmentStmt> → ID <whichStmt> | |
| | | <assignmentStmt>.addr=ID.addr |
| | <whichStmt>.inh_addr=<assignmentStmt>.addr | |
| | | <assignementStmt>.syn_addr=<whichStmt>.syn_addr |
| | | free(<whichStmt>) |
| 46 | <whichStmt> →<lvalueIDStmt> | |
| | <lvalueIDStmt>.inh_addr=<whichStmt>.inh_addr | |
| | | <whichStmt>.syn_addr=<lvalueIDStmt>,syn_addr |
| | | free(<lvalueIDStmt>) |
| 47 | <whichStmt> → <lvalueARRStmt> | |
| | <lvalueARRStmt>.inh_addr=<whichStmt>.inh_addr | |
| | | <whichStmt>.syn_addr=<lvalueARRStmt>,syn_addr |
| | | free(<lvalueARRStmt>) |
| 48 | <lvalueIDStmt> → ASSIGNOP <expression> SEMICOL | |
| | | <lvalueIDStmt>.syn_addr=createNewNode(label:lvalueIDStmt,<lvalueIDStmt>.inh_addr,<expression>.syn_addr) |

| | | |
|---|---|---|
| | | free(ASSIGNOP) |
| | | free(<expression>) |
| | | free(SEMICOL) |
| 49 | <lvalueARRStmt> → SQBO <exprIndex> SQBC ASSIGNOP <expression> SEMICOL | |
| | | <lvalueARRStmt>.addr=createNewNode(label:arrayElement,<lvalueARRStmt>.inh_addr,<exprIndex>.syn_addr) |
| | | <lvalueARRStmt>.syn_addr=createNewNode(label:lvalueARRStmt,<lvalueARRStmt>.addr,<expression>.syn_index) |
| | | free(SQBO) |
| | | free(SQBC) |
| | | free(exprIndex) |
| | | free(ASSIGNOP) |
| | | free(<expression>) |
| | | free(SEMICOL) |
| 50 | <arr_index> → <sign><index2> | |
| | | <arr_index>.addr=createNewNode(label:signedNum,<sign>.addr,<index2>.addr) |
| | | free(<sign>) |
| | | free(<index2>) |
| 51 | <index2> → ID | |
| | | <index2>.addr=ID.addr |
| 52 | <index2> → NUM | |
| | | <index2>.addr=NUM.addr |
| 53 | <sign> → ε | |
| | | <sign>.addr=NULL |
| 54 | <sign> → MINUS | |
| | | <sign>.addr=MINUS.addr |
| 55 | <sign> → PLUS | |
| | | <sign>.addr=PLUS.addr |
| 56 | <moduleReuseStmt> → <optional> USE MODULE ID WITH PARAMETERS <actual_para_list> SEMICOL | |
| | | <moduleReuseStmt>.addr=createNewNode(label:moduleReuseStmt,ID.addr,<actual_para_list>.syn_addr) |
| | <optional>.inh_addr=<moduleReuseStmt>.addr | |
| | | <moduleReuseStmt>.syn_addr=<optional>.syn_addr |
| | | free(<optional>) |
| | | free(USE) |
| | | free(MODULE) |
| | | free(WITH) |
| | | free(PARAMETERS) |
| | | free(<actual_para_list>) |
| | | free(SEMICOL) |
| 57 | <actual_para_list> → <sign> <K> <N9> | |
| | | <actual_para_list>.addr=createNewNode(label:signedValue,<sign>.addr,<K>.syn_addr) |
| | | <actual_para_list>.syn_addr=addToStart(<actual_para_list>.addr,<N9>.syn_addr) |
| | | free(<sign>) |
| | | free(<K>) |
| | | free(<N9>) |
| 58 | <N9(1)> → COMMA <sign> <K> <N9(2)> | |
| | | <N9(1)>.addr=createNewNode(label:signedValue,<sign>.addr,<K>.syn_addr) |
| | | <N9(1)>.syn_addr=addToStart(<N9(1)>.addr,<N9(2)>.syn_addr) |
| | | free(COMMA) |
| | | free(<sign>) |
| | | free(<K>) |
| | | free(<N9(2)>) |

| | | |
|---|---|---|
| 59 | <N9> → ε | |
| | | <N9>.syn_addr=NULL |
| 60 | <K> → NUM | |
| | | <K>.syn_addr = NUM.addr |
| 61 | <K> → RNUM | |
| | | <K>.syn_addr = RNUM.addr |
| 62 | <K> → <boolConst> | |
| | | <K>.syn_addr=<boolConst>.addr |
| | | free(<boolConst>) |
| 63 | <K> → ID <actual_para_list2> | |
| | | <K>.addr=ID.addr |
| | <actual_para_list2>.inh_addr=<K>.addr | |
| | | <K>.syn_addr=<actual_para_list2>.syn_addr |
| | | free(<actual_para_list2>) |
| 64 | <actual_para_list2> → SQBO <exprIndex> SQBC | |
| | | <actual_para_list2>.syn_addr=<exprIndex>.syn_addr |
| | | free(SQBO) |
| | | free(<exprIndex>) |
| | | free(SQBC) |
| 65 | <actual_para_list2> → ε | |
| | | <actual_para_list2>.syn_addr=<actual_para_list>.inh_addr |
| | | free(ε) |
| 66 | <optional> → SQBO <idList> SQBC ASSIGNOP | |
| | | <optional>.syn_addr=createNewNode(label:assignOp,<idList>.syn_addr,<optional>.inh_addr) |
| | | free(SQBO) |
| | | free(<idList>) |
| | | free(SQBC) |
| | | free(ASSIGNOP) |
| 67 | <optional> → ε | |
| | | <optional>.syn_addr=<optional>.inh_addr |
| | | free(ε) |
| 68 | <idList> -> ID <idList2> | |
| | | <idList>.syn_addr=addToStart(ID.addr,<idList2>.syn_addr) |
| | | free(<idList2>) |
| 69 | <idList2(1)> -> COMMA ID <idList2(2)> | |
| | | <idList2(1)>.syn_addr=addToStart(ID.addr,<idList2(2)>.syn_addr) |
| | | free(COMMA) |
| | | free(<idList2(2)>) |
| 70 | <idList2> -> ε | |
| | | <idList2>.syn_addr=NULL |
| | | free(ε) |
| 71 | <expression> -> <abExpr> | |
| | | <expression>.syn_addr=<abExpr>.syn_addr |
| | | free(<abExpr>) |
| 72 | <expression> -> <U> | |
| | | <expression>.syn_addr=<U>.syn_addr |
| | | free(<U>) |
| 73 | <U> -> <unary_op> <non_term> | |
| | | <U>.syn_addr=createNewNode(label:unaryOp,<unaryOp>.addr,<non_term>.syn_addr) |
| | | free(<unary_op>) |

|  |  |  |
|---|---|---|
|  |  | free(<non_term>) |
| 74 | <non_term> -> BO <arithmeticExpr> BC |  |
|  |  | <non_term>.syn_addr=<arithmeticExpr>.syn_addr |
|  |  | free(BO) |
|  |  | free(<arithmeticExpr>) |
|  |  | free(BC) |
| 75 | <non_term> ->  <var_const> |  |
|  |  | <non_term>.syn_addr=<var_const>.addr |
|  |  | free(<var_const>) |
| 76 | <unary_op> → PLUS |  |
|  |  | <unary_op>.addr = PLUS |
| 77 | <unary_op> → MINUS |  |
|  |  | <unary_op>.addr = MINUS |
| 78 | <abExpr> → <AnyTerm> <N7> |  |
|  |  | <abExpr>.addr=<AnyTerm>.syn_addr |
|  | <N7>.inh_addr=<abExpr>.addr |  |
|  |  | <abExpr>.syn_addr=<N7>.syn_addr |
|  |  | free(AnyTerm) |
|  |  | free(<N7>) |
| 79 | <N7(1)> → <logicalOp><AnyTerm><N7(2)> |  |
|  |  | <N7(1)>.addr=createNewNode(label:<logicalOp>,<N7(1)>.inh_addr,<AnyTerm>.syn_addr) |
|  | <N7(2)>.inh_addr=<N7(1)>.addr |  |
|  |  | <N7(1)>.syn_addr=<N7(2)>.syn_addr |
|  |  | free(<logicalOp>) |
|  |  | free(<AnyTerm>) |
|  |  | free(<N7(2)> |
| 80 | <N7> → ε |  |
|  |  | <N7>.syn_addr=<N7>.inh_addr |
|  |  | free(ε) |
| 81 | <AnyTerm> → <arithmeticExpr> <N8> |  |
|  |  | <AnyTerm>.addr=<arithmeticExpr>.syn_addr |
|  | <N8>.inh_addr=<AnyTerm>.addr |  |
|  |  | <AnyTerm>.syn_addr=<N8>.syn_addr |
|  |  | free(<arithmeticExpr>) |
|  |  | free(<N8>) |
| 82 | <AnyTerm> →  <boolConst> |  |
|  |  | <AnyTerm>.syn_addr=<boolConst>.addr |
|  |  | free(<boolConst>) |
| 83 | <N8> → <relationalOp> <arithmeticExpr> |  |
|  |  | <N8>.syn_addr=createNewNode(label:<relationalOp>,<N8>.inh_addr,<arithmeticExpr>.syn_addr) |
|  |  | free(<relationalOp>) |
|  |  | free(<arithmeticExpr>) |
| 84 | <N8> → ε |  |
|  |  | <N8>.syn_addr=<N8>.inh_addr |
|  |  | free(ε) |
| 85 | <var_const> → ID |  |
|  |  | <var_const>.addr = ID |
| 86 | <var_const> → NUM |  |
|  |  | <var_const>.addr = NUM |
| 87 | <var_const> → RNUM |  |

| | | |
|---|---|---|
| | | <var_const>.addr = RNUM |
| 88 | <arithmeticExpr> → <term>  <arithmeticExpr2> | |
| | | <arithmeticExpr>.addr=<term>.syn_addr |
| | <arithmeticExpr2>.inh_addr=<arithmeticExpr>.addr | |
| | | <arithmeticExpr>.syn_addr=<arithmeticExpr2>.syn_addr |
| | | free(<term>) |
| | | free(<arithmeticExpr2>) |
| 89 | <arithmeticExpr2(1)> → <low_op> <term>  <arithmeticExpr2(2)> | |
| | | <arithmeticExpr2(1)>.addr=createNewNode(label:<low_op>,<arithmeticExpr2(1)>..inh_addr,<term>.syn_addr) |
| | <arithmeticExpr2(2)>.inh_addr=<arithmeticExpr2(1)>.addr | |
| | | <arithmeticExpr2(1)>.syn_addr=<arithmeticExpr2(2)>.syn_addr |
| | | free(<low_op>) |
| | | free(<term>) |
| | | free(<arithmeticExpr2(2)>) |
| 90 | <arithmeticExpr2> →  ε | |
| | | <arithmeticExpr2>.syn_addr=<arithmeticExpr2>.inh_addr |
| 91 | <term> → <factor> <term2> | |
| | | <term>.addr=<factor>.syn_addr |
| | <term2>.inh_addr=<term>.addr | |
| | | <term>.syn_addr=<term2>.syn_addr |
| | | free(<factor>) |
| | | free(<term2>) |
| 92 | <term2> →  ε | |
| | | <term2>.syn_addr=<term2>.inh_addr |
| 93 | <term2(1)> → <high_op> <factor> <term2(2)> | |
| | | <term2(1)>.addr=createNewNode(label:<high_op>,<term2(1)>.inh_addr,<factor>.syn_addr) |
| | <term2(2)>.inh_addr=<term2(1)>.addr | |
| | | <term2(2)>.syn_addr=<term2(1)>.syn_addr |
| | | free(<high_op>) |
| | | free(<factor>) |
| | | free(<term2(2)>) |
| 94 | <factor> → BO <abExpr> BC | |
| | | <factor>.syn_addr=<abExpr>.syn_addr |
| | | free(BO) |
| | | free(<abExpr>) |
| | | free(BC) |
| 95 | <factor> -> NUM | |
| | | <factor>.syn_addr=NUM.addr |
| 96 | <factor> -> RNUM | |
| | | <factor>.syn_addr=RNUM.addr |
| 97 | <factor> ->  <boolConst> | |
| | | <factor>.syn_addr=<boolConst>.addr |
| | | free(<boolConst>) |
| 98 | <factor> ->  ID <factor2> | |
| | | <factor>.addr=ID.addr |
| | <factor2>.inh_addr=<factor>.addr | |
| | | <factor>.syn_addr=<factor2>.syn_addr |
| | | free(ID) |
| | | free(<factor2>) |
| 99 | <factor2> -> ε | |

| | | |
|---|---|---|
| | | <factor2>.syn_addr=<factor2>.inh_addr |
| 100 | <factor2> -> SQBO <exprIndex> SQBC | |
| | | <factor2>.syn_addr=createNewNode(label:arrayExprIndex,<factor2>.inh_addr,<exprIndex>.syn_addr) |
| | | free(SQBO) |
| | | free(SQBC) |
| | | free(<exprIndex>) |
| 101 | <exprIndex> → <sign> <exprIndex2> | |
| | | <exprIndex>.syn_addr=createNewNode(label:signedExprIndex,<sign>.addr,<exprIndex2>.syn_addr) |
| | | free(<sign>) |
| | | free(<exprIndex2>) |
| 102 | <exprIndex> → <arrExpr> | |
| | | <exprIndex>.syn_addr=<arrExpr>.syn_addr |
| | | free(<arrExpr>) |
| 103 | <exprIndex2> → <index2> | |
| | | <exprIndex2>.syn_addr=<index2>.syn_addr |
| | | free(<index2>) |
| 104 | <exprIndex2> → BO <arrExpr> BC | |
| | | <exprIndex2>.syn_addr=<arrExpr>.syn_addr |
| | | free(BO) |
| | | free(<arrExpr>) |
| | | free(BC) |
| 105 | <arrExpr> → <arrTerm> <arrExpr2> | |
| | | <arrExpr>.addr=<arrTerm>.syn_addr |
| | <arrExpr2>.inh_addr=<arrExpr>.addr | |
| | | <arrExpr>.syn_addr=<arrExpr2>.syn_addr |
| | | free(<arrTerm>) |
| | | free(<arrExpr2>) |
| 106 | <arrExpr2> → ε | |
| | | <arrExpr2>.syn_addr=<arrExpr2>.inh_addr |
| 107 | <arrExpr2(1)> → <low_op> <arrTerm> <arrExpr2(2)> | |
| | | <arrExpr2(1)>.addr=createNewNode(label:<low_op>,<arrExpr2(1)>.inh_addr,<arrTerm>.syn_addr) |
| | <arrExpr2(2)>.inh_addr=<arrExpr2(1)>.addr | |
| | | <arrExpr2(1)>.syn_addr=<arrExpr2(2)>.syn_addr |
| | | free(<low_op>) |
| | | free(<arrTerm>) |
| | | free(<arrExpr2(2)>) |
| 108 | <arrTerm> → <arrFactor> <arrTerm2> | |
| | | <arrTerm>.addr=<arrFactor>.syn_addr |
| | <arrTerm2>.inh_addr=<arrTerm>.addr | |
| | | <arrTerm>.syn_addr=<arrTerm2>.syn_addr |
| | | free(<arrFactor>) |
| | | free(<arrTerm2>) |
| 109 | <arrTerm2(1)> → <high_op> <arrFactor> <arrTerm2(2)> | |
| | | <arrTerm2(1)>.addr=createNewNode(label:<high_op>,<arrTerm2(1)>.inh_addr,<arrFactor>.syn_addr) |
| | <arrTerm2(2)>.inh_addr=<arrTerm2(1)>.addr | |
| | | <arrTerm2(1)>.syn_addr=<arrTerm2(2)>,syn_addr |
| | | free(<high_op>) |
| | | free(<arrFactor>) |
| | | free(<arrTerm2(2)>) |
| 110 | <arrTerm2> → ε | |

| | | |
|---|---|---|
| | | <arrTerm2>.syn_addr=<arrTerm2>.inh_addr |
| 111 | <arrFactor> → ID | |
| | | <arrFactor>.syn_addr=ID.addr |
| 112 | <arrFactor> → NUM | |
| | | <arrFactor>.syn_addr=NUM.addr |
| 113 | <arrFactor> → <boolConst> | |
| | | <arrFactor>.syn_addr=<boolConst>.addr |
| | | free(<boolConst>) |
| 114 | <arrFactor> → BO <arrExpr> BC | |
| | | <arrFactor>.syn_addr=<arrExpr>.syn_addr |
| | | free(BO) |
| | | free(BC) |
| | | free(<arrExpr>) |
| 115 | <low_op> → PLUS | |
| | | <low_op>.addr = PLUS.addr |
| 116 | <low_op> → MINUS | |
| | | <low_op>.addr = MINUS.addr |
| 117 | <high_op> → MUL | |
| | | <high_op>.addr = MUL.addr |
| 118 | <high_op> → DIV | |
| | | <high_op>.addr = DIV.addr |
| 119 | <logicalOp>→ OR | |
| | | <logicalOp>.addr = OR.addr |
| 120 | <logicalOp>→ AND | |
| | | <logicalOp>.addr = AND.addr |
| 121 | <relationalOp>→ LT | |
| | | <relationalOp>.addr = LT.addr |
| 122 | <relationalOp>→ LE | |
| | | <relationalOp>.addr = LE.addr |
| 123 | <relationalOp>→ GT | |
| | | <relationalOp>.addr = GT.addr |
| 124 | <relationalOp>→ GE | |
| | | <relationalOp>.addr = GE.addr |
| 125 | <relationalOp>→ EQ | |
| | | <relationalOp>.addr = EQ.addr |
| 126 | <relationalOp>→ NE | |
| | | <relationalOp>.addr = NE.addr |
| 127 | <declareStmt>→ DECLARE <idList> COLON <dataType> SEMICOL | |
| | | <declareStmt>.addr=createNewNode(label:declareStmt,<idList>.syn_addr,<dataType>.addr) |
| | | free(DECLARE) |
| | | free(<idList>) |
| | | free(COLON) |
| | | free(<dataType>) |
| | | free(SEMICOL) |
| 128 | <conditionalStmt>→ SWITCH BO ID BC START <caseStmts><default_stmt> END | |
| | | <conditionalStmt>.addr=createNewNode(label:switchStmt,ID.addr,<caseStmts>.syn_addr,<default_Stmt>.syn_addr) |
| | | free(SWITCH) |
| | | free(BO) |
| | | free(BC) |
| | | free(START) |

| | | |
|---|---|---|
| | | free(<caseStmts>) |
| | | free(<defaultStmts>) |
| | | free(END) |
| 129 | <caseStmts>→ CASE <value> COLON <statements> BREAK SEMICOL <caseStmts2> | |
| | | <caseStmts>.addr=createNewNode(label:caseValueStmts,<value>.addr,<statements>.syn_addr) |
| | | <caseStmts>.syn_addr=addToStart(<caseStmts>.addr,<caseStmts2>.syn_addr) |
| | | free(CASE) |
| | | free(<value>) |
| | | free(COLON) |
| | | free(<statements>) |
| | | free(BREAK) |
| | | free(SEMICOL) |
| | | free(<caseStmts2>) |
| 130 | <caseStmts2(1)>→ CASE <value> COLON <statements> BREAK SEMICOL <caseStmts2(2)> | |
| | | <caseStmts2(1)>.addr=createNewNode(label:caseValueStmts,<value>.addr,<statements>.syn_addr) |
| | | <caseStmts2(1)>.syn_addr=addToStart(<caseStmts2(1)>.addr,<caseStmts2(2)>.syn_addr) |
| | | free(CASE) |
| | | free(<value>) |
| | | free(COLON) |
| | | free(<statements>) |
| | | free(BREAK) |
| | | free(SEMICOL) |
| | | free(<caseStmts2(2)>) |
| 131 | <caseStmts2>→ ε | |
| | | <caseStmts2>.syn_addr=NULL |
| | | free(ε) |
| 132 | <value> → NUM | |
| | | <value>.addr = NUM.addr |
| 133 | <value> → TRUE | |
| | | <value>.addr = TRUE.addr |
| 134 | <value> → FALSE | |
| | | <value>.addr = FALSE.addr |
| 135 | <default_stmt> →DEFAULT COLON <statements> BREAK SEMICOL | |
| | | <default_stmt>.syn_addr=createNewNode(label:defaultStmt.<statements>.syn_addr) |
| | | free(DEFAULT) |
| | | free(COLON) |
| | | free(<statements>) |
| | | free(BREAK) |
| | | free(SEMICOL) |
| 136 | <default_stmt> → ε | |
| | | <default_stmt>.syn_addr=NULL |
| | | free(ε) |
| 137 | <iterativeStmt>→ FOR BO ID IN <for_range> BC START <statements> END | |
| | | <iterativeStmt>.addr=createNewNode(label:forloop,<for_range>.addr,<statements>.syn_addr) |
| | | free(FOR) |
| | | free(BO) |
| | | free(IN) |
| | | free(<for_range>) |
| | | free(BC) |
| | | free(START) |

| | | |
|---|---|---|
| | | free(<statements>) |
| | | free(END) |
| 138 | <iterativeStmt>→  WHILE BO <abExpr> BC START <statements> END | |
| | | <iterativeStmt>.addr= createNewNode(label:whileloop,<abExpr>.syn_addr,<statements>.syn_addr) |
| | | free(WHILE) |
| | | free(BO) |
| | | free(<abExpr>) |
| | | free(BC) |
| | | free(START) |
| | | free(<statements>) |
| | | free(END) |
| 139 | <for_range> → <for_index(1)> RANGEOP <for_index(2)> | |
| | | <for_range>.addr=createNewNode(label:rangeOP,<for_index(1)>.addr,<for_index(2)>.addr) |
| | | free(<for_index(1)>) |
| | | free(<for_index(2)>) |
| | | free(<RANGEOP>) |
| 140 | <for_index> → <for_sign> <for_index2> | |
| | | <for_index>.addr=createNewNode(label:signedNum,<for_sign>.addr,<for_index2>.addr) |
| | | free(<for_sign>) |
| | | free(<for_index2) |
| 141 | <for_index2> → NUM | |
| | | <for_index2>.addr = NUM.addr |
| 142 | <for_sign> →  ε | |
| | | <for_sign>.addr=NULL |
| | | free(ε) |
| 143 | <for_sign> → MINUS | |
| | | <for_sign>.addr=MINUS.addr |
| 144 | <for_sign> → PLUS | |
| | | <for_sign>.addr=PLUS.addr |