# ABSTRACT SYNTAX TREE FORMATION RULES

Done By : Group No. 16 :-

Gandhi Atith Nikeshkumar (2017A7PS0062P)

Burhan Boxwalla (2017A7PS0097P)

Keshav Sharma (2017A7PS0140P)

Shray Mathur (2017A7PS1180P)

Raj Sanjay Shah (2017A7PS1181P)

| S. No | Grammar Rule | Abstract Syntax Tree Formation Rule |
|---|---|---|
| 1) | <program> → <moduleDeclarations> <otherModules> <driverModule> <otherModules> | <program>.addr= make_Node(<moduleDeclaration>.addr, <otherModules>1.addr, <driverModule>.addr, <otherModules>2.addr) |
| 2) | <moduleDeclarations> → <moduleDeclaration> <moduleDeclarations> | <moduleDeclarations>1.addr= make_Node(<moduleDeclaration>.addr, <moduleDeclarations>2.addr) |
| 3) | <moduleDeclarations> → ε | <moduleDeclarations>.addr= NULL |
| 4) | <moduleDeclaration> → DECLARE MODULE ID SEMICOL | ID.addr = make_Leaf(MODULE, ID.Lexeme)<br><moduleDeclaration>.addr = ID.addr |
| 5) | <otherModules> → <module> <otherModules> | <otherModules>1.addr = make_Node (<module>.addr, <otherModules>2.addr) |
| 6) | <otherModules> → ε | <otherModules>.addr = NULL |
| 7) | <driverModule> → DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef> | <driverModule>.addr = <moduleDef>.addr |
| 8) | <module> → DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef> | ID.addr = make_Leaf(MODULE, ID.Lexeme)<br><module>.addr = make_Node( ID.addr, <input_list>.addr, <ret>.addr, <moduleDef>.addr) |
| 9) | <ret> → RETURNS SQBO <output_plist> SQBC SEMICOL | <ret>.addr = <output_list>.addr |
| 10) | <ret> → ε | <ret>.addr = NULL |
| 11) | <input_plist> → ID COLON <dataType> <iplist> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><input_list>.addr = make_Node(ID.addr, <dataType>.addr, <iplist>.addr) |
| 12) | <iplist> → COMMA ID COLON <dataType> <iplist> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><iplist>1.addr = make_Node(ID.addr, <dataType>.addr, <iplist>2.addr) |
| 13) | <iplist>→ ε | <iplist>.addr = NULL |
| 14) | <output_plist> → ID COLON <type> <oplist> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><output_plist>.addr = make_Node(ID.addr, <type>.addr, <oplist>.addr) |
| 15) | <oplist> → COMMA ID COLON <type> <oplist> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><oplist>1.addr = make_Node (ID.addr, <type>.addr, <oplist>2.addr) |
| 16) | <oplist> → ε | <oplist>.addr= NULL |
| 17) | <dataType> → INTEGER | <dataType>.type = Integer |
| 18) | <dataType> → REAL | <dataType>.type = Real |
| 19) | <dataType> → BOOLEAN | <dataType>.type = Boolean |
| 20) | <dataType> → ARRAY SQBO <dynamic_range> SQBC OF <type> | <dataType>.addr = make_Node(<dynamic_range>.addr, <type>.addr) |
| 21) | <dynamic_range>→<index> RANGEOP <index> | <dynamic_range>.addr = make_Node (<index>1.addr, <index>2.addr) |
| 22) | <type> → INTEGER | <type>.type = Integer |
| 23) | <type> → REAL | <type>.type = Real |

| S. No | Grammar Rule | Abstract Syntax Tree Formation Rule |
|---|---|---|
| 24) | <type> → BOOLEAN | <type>.type = Boolean |
| 25) | <moduleDef> → START <statements> END | <moduleDef>.addr = <statements>.addr |
| 26) | <statements> → <statement> <statements> | <statements>1 .addr = make_Node(<statement>.addr, <statements>2 .addr) |
| 27) | <statements> → ε | <statements>.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> → <condionalStmt> | <statement>.addr = <condionalStmt>.addr |
| 32) | <statement> → <iterativeStmt> | <statement>.addr = <iterativeStmt>.addr |
| 33) | <ioStmt> → GET_VALUE BO ID BC SEMICOL | ID.addr = make_Leaf(ID , ID.Lexeme)<br>GET_VALUE.addr = make_Leaf(GET_VALUE, GET_VALUE.Lexeme)<br><ioStmt>.addr = make_Node(GET_VALUE.addr , ID.addr) |
| 34) | <ioStmt> → PRINT BO <print> BC SEMICOL | PRINT.addr = make_Leaf(PRINT, PRINT.Lexeme)<br><ioStmt>.addr = make_Node(PRINT.addr, <print>.addr) |
| 35) | <print> → <var> | <print>.addr = <var>.addr |
| 36) | <print> → <boolConst> | <print>.addr = <boolConst>.addr |
| 37) | <var> → ID <whichId> | ID.addr = make_Leaf(ID , ID.Lexeme)<br><var>.addr = make_Node(ID.addr, <whichId>.addr) |
| 38) | <var> → NUM | NUM.addr = make_Leaf(NUM, NUM.value)<br><var>.addr = NUM.addr |
| 39) | <var> → RNUM | RNUM.addr = make_Leaf(RNUM, RNUM.value)<br><var>.addr = RNUM.addr |
| 40) | <whichId> → SQBO <index> SQBC | <whichId>.addr = <index>.addr |
| 41) | <whichId> → ε | <whichId>.addr = NULL |
| 42) | <simpleStmt> → <assignmentStmt> | <simpleStmt>.addr = <assignmentStmt>.addr |
| 43) | <simpleStmt> → <moduleReuseStmt> | <simpleStmt>.addr = <moduleReuseStmt>.addr |
| 44) | <assignmentStmt> → ID <whichStmt> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><assignmentStmt>.addr = make_Node(ID.addr, <whichStmt>.addr) |
| 45) | <whichStmt> → <lvalueIDStmt> | <whichStmt>.addr = <lvalueIDStmt>.addr |
| 46) | <whichStmt> → <lvalueARRStmt> | <whichStmt>.addr = <lvalueARRStmt>.addr |

| S. No | Grammar Rule | Abstract Syntax Tree Formation Rule |
|---|---|---|
| 47) | <lvalueIDStmt> → ASSIGNOP <expression_new> SEMICOL | ASSIGNOP.addr = make_Leaf(ASSIGNOP, ":=")<br><lvalueIDStmt>.addr = make_Node(ASSIGNOP.addr, <expression_new>.addr) |
| 48) | <lvalueARRStmt> → SQBO <index> SQBC ASSIGNOP <expression_new> SEMICOL | ASSIGNOP.addr = make_Leaf(ASSIGNOP, ":=")<br><lvalueARRStmt>.addr = make_Node(<index>.addr, ASSIGNOP.addr, <expression_new>.addr) |
| 49) | <index> → NUM | NUM.addr = make_Leaf(NUM, NUM.value)<br><index>.addr = NUM.addr |
| 50) | <index> → ID | ID.addr = make_Leaf(ID, ID.Lexeme)<br><index>.addr = ID.addr |
| 51) | <moduleReuseStmt> → <optional> USE MODULE ID WITH PARAMETERS <idList> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><moduleReuseStmt>.addr = make_Node(<optional>.addr, ID.addr, <idList>.addr) |
| 52) | <optional> →SQBO <idList> SQBC ASSIGNOP | <optional>.addr = <idList>.addr |
| 53) | <optional> → ε | <optional>.addr = NULL |
| 54) | <idList> → ID <idLists> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><idList>.addr = make_Node(ID.addr, <idLists>.addr) |
| 55) | <idLists> →COMMA ID <idLists> | ID.addr = make_Leaf(ID, ID.Lexeme)<br><idLists>1.addr = make_Node(ID.addr, <idLists>2.addr) |
| 56) | <idLists> → ε | <idLists>.addr = NULL |
| 57) | <expression_new> → <U> | <expression_new>.addr = <U>.addr |
| 58) | <expression_new> → <arithmeticOrBooleanExpression> | <expression_new>.addr = <arithmenticOrBooleanExpression>.addr |
| 59) | <U> → <op1> <U'> | <U>.addr = make_Node(<op1.addr>, <U'>.addr) |
| 60) | <U'> → BO <arithmeticExpr> BC | <U'>.addr = <arithmeticExpr>.addr |
| 61) | <U'> → <var> | <U'>.addr = <var>.addr |
| 62) | <arithmeticOrBooleanExpression> → <boolTerm> <followingBool> | <arithmeticOrBooleanExpr>.addr = make_Node(<boolTerm>.addr, <followingBool>.addr) |
| 63) | <followingBool> → <logicalOp> <boolTerm> <followingBool> | <followingBool>1.addr = make_Node(<logicalOp>.addr, <boolTerm>.addr, <followingBool>2.addr) |
| 64) | <followingBool> → ε | <followingBool>.addr = NULL |
| 65) | <boolTerm> → <arithmeticExpr> <boolean> | <boolTerm>.addr = make_Node(<arithmeticExpr>.addr, <boolean>.addr) |
| 66) | <boolTerm> → <boolConst> | <boolTerm>.addr = <booConst>.addr |
| 67) | <boolean> → <relationalOp> <arithmeticExpr> | <boolean>.addr = make_Node(<relationalOp>.addr, <arithmeticExpr>.addr) |
| 68) | <boolean> → ε | <boolean>.addr = NULL |
| 69) | <arithmeticExpr> → <term> <followingArithExp> | <arithmeticExpr>.addr = make_Node(<term>.addr, <followingArithExp>.addr) |

| S. No | Grammar Rule | Abstract Syntax Tree Formation Rule |
|---|---|---|
| 70) | <followingArithExp> → <op1> <term> <followingArithExp> | <followingArithExp>.addr = make_Node(<op1>.addr, <term>.addr, <followingArithExp>.addr) |
| 71) | <followingArithExp> → ε | <followingArithExp>.addr = NULL |
| 72) | <term> → <factor> <followingTerm> | <term>.addr = make_Node(<factor>.addr, <followingTerm>.addr) |
| 73) | <followingTerm> → <op2> <factor> <followingTerm> | <followingTerm>.addr = make_Node(<op2>.addr, <factor>.addr, <followingTerm>.addr) |
| 74) | <followingTerm> → ε | <followingTerm>.addr = NULL |
| 75) | <factor> → BO <arithmeticOrBooleanExpression> BC | <factor>.addr = <arithmeticOrBooleanExpression>.addr |
| 76) | <factor> → <var> | <factor>.addr = <var>.addr |
| 77) | <op1> → PLUS | PLUS.addr = make_Leaf(PLUS, '+')<br><op1>.addr = PLUS.addr |
| 78) | <op1> → MINUS | MINUS.addr = make_Leaf(MINUS, '-')<br><op1>.addr = MINUS.addr |
| 79) | <op2> → MUL | MUL.addr = make_Leaf(MUL, '*')<br><op2>.addr = MUL.addr |
| 80) | <op2> → DIV | DIV.addr = make_Leaf(DIV, '/')<br><op2>.addr = DIV.addr |
| 81) | <logicalOp> → AND | AND.addr = make_Leaf(AND, 'AND')<br><logicalOp>.addr = AND.addr |
| 82) | <logicalOp> → OR | OR.addr = make_Leaf(OR, 'OR')<br><logicalOp>.addr = OR.addr |
| 83) | <relationalOp> → LT | LT.addr = make_Leaf(LT, '<')<br><relationalOp>.addr = LT.addr |
| 84) | <relationalOp> → LE | LE.addr = make_Leaf(LE, '<=)<br><relationalOp>.addr = LE.addr |
| 85) | <relationalOp> → GT | GT.addr = make_Leaf(GT, '>')<br><relationalOp>.addr = GT.addr |
| 86) | <relationalOp> → GE | GE.addr = make_Leaf(GE, '>=')<br><relationalOp>.addr = GE.addr |
| 87) | <relationalOp> → EQ | EQ.addr = make_Leaf(EQ, '==')<br><relationalOp>.addr = EQ.addr |
| 88) | <relationalOp> → NE | NE.addr = make_Leaf(NE, '!=')<br><relationalOp>.addr = NE.addr |
| 89) | <declareStmt> → DECLARE <idList> COLON <dataType> SEMICOL | <declareStmt>.addr = make_Node(<idList>.addr, <dataType>.addr) |
| 90) | <condionalStmt> → SWITCH BO ID BC START <caseStmts> <default> END | <conditionalStmt>.addr = make_Node(<caseStmts>.addr, <default>.addr) |
| 91) | <caseStmts> → CASE <value> COLON <statements> BREAK SEMICOL <caseStmt> | <caseStmts>.addr = make_Node(<value>.addr, <statements>.addr, <caseStmt>.addr) |
| 92) | <caseStmt> → CASE <value> COLON <statements> BREAK SEMICOL <caseStmt> | <caseStmt>.addr = make_Node(<value>.addr, <statements>.addr, <caseStmt>.addr) |

| S. No | Grammar Rule | Abstract Syntax Tree Formation Rule |
|---|---|---|
| 93) | <caseStmt> → ε | <caseStmt>.addr = NULL |
| 94) | <value> → NUM | NUM.addr = make_Leaf(NUM, Num.value)<br><value>.addr = NUM.addr |
| 95) | <value> → <boolConst> | <value>.addr=<boolConst>.addr |
| 96) | <default> → DEFAULT COLON <statements> BREAK SEMICOL | <default>.addr = <statements>.addr |
| 97) | <default> → ε | <default>.addr = NULL |
| 98) | <iterativeStmt> → FOR BO ID IN <range> BC START <statements> END | ID.addr = make_Leaf(ID, ID.Lexeme)<br><iterativeStmt>.addr = make_Node(ID.addr, <range>.addr,<statements>.addr) |
| 99) | <iterativeStmt> → WHILE BO <arithmeticOrBooleanExpression> BC START <statements> END | <iterativeStmt>.addr = make_Node(<arithmeticOrBooleanExpression>.addr, <statements>.addr) |
| 100) | <range> → NUM RANGEOP NUM | NUM1.addr=make_Leaf(NUM1,NUM1.value)<br>NUM2.addr=make_Leaf(NUM2, NUM2.value)<br><range>.addr=make_Node(NUM1.addr,NUM2.addr) |
| 101) | <boolConst> → TRUE | TRUE.addr = make_Leaf(TRUE, 'TRUE')<br><boolConst>.addr = TRUE.addr |
| 102) | <boolConst> → FALSE | FALSE.addr = make_Leaf(FALSE, 'FALSE')<br><boolConst>.addr = FALSE.addr |