**Semantic Rules for Abstract Syntax Tree Generation**

Submitted By

**Naveen Venkat**

**2015A7PS0078P**

Corresponding to every grammar rule (numbered), the semantic rules have been written in sequence each within curly braces. A C-like pseudo code is followed to represent the corresponding semantic rules. Non terminal nodes are within angular brackets and terminal nodes are in capital letters.  
 **Attribute descriptions are as follows:**

* **carry :** used as a synthesized attribute
* **inh :** used as an inherited attribute
* **addr :** denotes theaddress of the corresponding parse tree node
* **link :** used to link the node with a sibling node (as in the case of a linked list)

Each of the above four attributes can be viewed as pointers pointing to nodes.

**Semantic rules are as follows:**

1. <mainFunction> ===> MAIN SQO SQC <stmtsAndFunctionDefs> END

{free(MAIN)}, {free(SQO)}, {free(SQC)}, {free(END)}, {<mainFunction> = newnode("MAINFUNCTION", <stmtsAndFunctionDefs>.addr)}

1. <stmtsAndFunctionDefs> ===> <stmtOrFunctionDef> <other1>

{<stmtOrFunctionDef>.carry.link = <other1>.carry}, {<stmtsAndFunctionDefs> = newnode("SAFD", <stmtOrFunctionDef>.carry)}, {free(<other1>)}, {free(<stmtsAndFunctionDefs>)}

1. <other1> ===> \_epsilon\_

{<other1>.carry=NULL}, {free(\_epsilon\_)}

1. <other1> ===> <stmtOrFunctionDef> <other1>

{<other1>\_1.carry = <stmtOrFunctionDef>.carry}, {<stmtOrFunctionDef>.carry.link = <other1>\_2.carry}, {free(<other1>\_2)}, {free(<stmtOrFunctionDef>)}

1. <stmtOrFunctionDef> ===> <stmt>

{<stmtOrFunctionDef>.carry = <stmt>.carry}, {free(<stmt>)}

1. <stmtOrFunctionDef> ===> <functionDef>

{<stmtOrFunctionDef>.carry = <functionDef>.addr}

1. <stmt> ===> <declarationStmt>

{<stmt>.carry = <declarationStmt>.addr}

1. <stmt> ===> <assignmentStmt\_type1>

{<stmt>.carry = <assignmentStmt\_type1>.addr}

1. <stmt> ===> <assignmentStmt\_type2>

{<stmt>.carry = <assignmentStmt\_type2>.addr}

1. <stmt> ===> <ifStmt>

{<stmt>.carry = <ifStmt>.addr}

1. <stmt> ===> <ioStmt>

{<stmt>.carry = <ioStmt>.addr}

1. <stmt> ===> <funCallStmt> SEMICOLON

{<stmt>.carry = <funCallStmt>.addr}, {free(SEMICOLON)}

1. <functionDef> ===> FUNCTION SQO <parameter\_list> SQC ASSIGNOP FUNID SQO <parameter\_list> SQC <stmtsAndFunctionDefs> END SEMICOLON

{free(FUNCTION)}, {free(SQO\_1)}, {free(SQC\_1)}, {free(ASSIGNOP)}, {free(SQO\_2)}, {free(SQC\_2)}, {free(END)}, {free(SEMICOLON)}, {<functionDef> = newnode(<parameter\_list>\_1.addr, FUNID.addr, <parameter\_list>\_2.addr, <stmtsAndFunctionDefs>.addr)}

1. <parameter\_list> ===> <type> ID <remainingList>

{<type>.link = ID.addr}, {ID.link = <remainingList>.carry}, {free(<remainingList>)}, {<parameter\_list> = newnode("PL", <type>.addr)}

1. <remainingList> ===> COMMA <type> ID <remainingList>

{free(COMMA)}, {free(ID)}, {<type>.link = ID.addr}, {ID.link = <remainingList>\_2.carry}, {<remainingList>\_1.carry = <type>.addr}, {free(<remainingList>\_2)}

1. <remainingList> ===> \_epsilon\_

{<remainingList>.carry = NULL}, {free(\_epsilon\_)}

1. <type> ===> INT

{<type>.carry = INT.addr}

1. <type> ===> REAL

{<type>.carry = REAL.addr}

1. <type> ===> STRING

{<type>.carry = STRING.addr}

1. <type> ===> MATRIX

{<type>.carry = MATRIX.addr}

1. <declarationStmt> ===> <type> <var\_list> SEMICOLON

{free(SEMICOLON)}, {<declarationStmt> = newnode("DEC", <type>.carry, <var\_list>.carry)}, {free(<var\_list>)}, {free(<type>)}

1. <var\_list> ===> ID <more\_ids>

{ID.link = <more\_ids>.carry}, {<var\_list>.carry = ID.addr}, {free(<more\_ids>)}

1. <more\_ids> ===> COMMA ID <more\_ids>

{ID.link = <more\_ids>.carry}, {<more\_ids>.carry = ID.addr}, {free(COMMA)}, {free(<more\_ids>)}

1. <more\_ids> ===> \_epsilon\_

{<more\_ids>.carry = NULL}, {free(\_epsilon\_)}

1. <assignmentStmt\_type1> ===> <leftHandSide\_singleVar> ASSIGNOP <rightHandSide\_type1> SEMICOLON

{free(SEMICOLON)}, {free(ASSIGNOP)}, {<assignmentStmt\_type1> = newnode("=1", <leftHandSide\_singleVar>.carry, <rightHandSide\_type1>.carry)}, {free(<leftHandSide\_singleVar>)}, {free(<rightHandSide\_type1>)}

1. <assignmentStmt\_type2> ===> <leftHandSide\_listVar> ASSIGNOP <rightHandSide\_type2> SEMICOLON

{free(SEMICOLON)}, {free(ASSIGNOP)}, {<assignmentStmt\_type2> = newnode("=2", <leftHandSide\_listVar>.carry, <rightHandSide\_type2>.carry)}, {free(<leftHandSide\_listVar>)}, {free(<rightHandSide\_type2>)}

1. <leftHandSide\_singleVar> ===> ID

{<leftHandSide\_singleVar>.carry = ID.addr}

1. <leftHandSide\_listVar> ===> SQO <var\_list> SQC

{free(SQO)}, {free(SQC)}, {<leftHandSide\_listVar>.carry = newnode(<var\_list>.carry)}, {free(<var\_list>)}

1. <rightHandSide\_type1> ===> <arithmeticExpression>

{<rightHandSide\_type1>.carry = <arithmeticExpression>.carry}, {free(<arithmeticExpression>)}

1. <rightHandSide\_type1> ===> <sizeExpression>

{<rightHandSide\_type1>.carry = <sizeExpression>.addr}

1. <rightHandSide\_type1> ===> <funCallStmt>

{<rightHandSide\_type1>.carry = <funCallStmt>.addr}

1. <rightHandSide\_type2> ===> <sizeExpression>

{<rightHandSide\_type2>.carry = <sizeExpression>.addr}

1. <rightHandSide\_type2> ===> <funCallStmt>

{<rightHandSide\_type2>.carry = <funCallStmt>.addr}

1. <sizeExpression> ===> SIZE ID

{free(SIZE)}, {<sizeExpression> = newnode("SIZE", ID.addr)}

1. <ifStmt> ===> IF OP <booleanExpression> CL <stmt> <otherStmts> <other2>

{free(IF)}, {free(OP)}, {free(CL)}, {<stmt>.link = <otherStmts>.carry}, {<ifStmt> = newnode("IF", <booleanExpression>.addr, <stmt>.carry, <other2>.carry)}, {free(<stmt>)}, {free(<other2>)}

1. <other2> ===> ELSE <stmt> <otherStmts> ENDIF SEMICOLON

{free(ELSE)}, {free(ENDIF)}, {free(SEMICOLON)}, {<stmt>.carry.link = <otherStmts>.carry}, {<other2>.carry = newnode("ELSE", <stmt>.carry)}, {free(<stmt>)}, {free(<otherStmts>)}

1. <other2> ===> ENDIF SEMICOLON

{<other2>.carry = NULL}, {free(ENDIF)}, {free(SEMICOLON)}

1. <otherStmts> ===> <stmt> <otherStmts>

{<otherStmts>\_1.carry = <stmt>}, {<stmt>.link = <otherStmts>\_2.carry}, {free(<otherStmts>\_2)}

1. <otherStmts> ===> \_epsilon\_

{<otherStmts>.carry = NULL}

1. <ioStmt> ===> READ OP ID CL SEMICOLON

{free(READ)}, {free(OP)}, {free(CL)}, {free(SEMICOLON)}, {<ioStmt> = newnode("IO\_READ", ID.addr)}

1. <ioStmt> ===> PRINT OP ID CL SEMICOLON

{free(PRINT)}, {free(OP)}, {free(CL)}, {free(SEMICOLON)}, {<ioStmt> = newnode("IO\_PRINT", ID.addr)}

1. <funCallStmt> ===> FUNID OP <inputParameterList> CL

{free(OP)}, {free(CL)}, {<funCallStmt> = newnode("CALL", FUNID.addr, <inputParameterList>.addr)

1. <inputParameterList> ===> <var> <listVar>

{<var>.carry.link = <listVar>.carry}, {free(<var>)}, {free(<listVar>)}

1. <inputParameterList> ===> \_epsilon\_

{<inputParameterList>.carry = NULL}, {free(\_epsilon\_)}

1. <listVar> ===> \_epsilon\_

{<listVar>.carry = NULL}, {free(\_epsilon\_)}

1. <listVar> ===> COMMA <var> <listVar>

{free(COMMA)}, {<var>.carry.link = <listVar>\_2.carry}, {free(<listVar>\_2)}, {free(<var>)}

1. <arithmeticExpression> ===> <arithmeticTerm> <other3>

{<other3>.inh = <arithmeticTerm>.carry}, {<arithmeticTerm>.carry = <other3>.carry}, {free(<other3>)}

1. <other3> ===> <operator\_lowPrecedence> <arithmeticTerm> <other3>

{<other3>\_2.inh = <arithmeticTerm>.carry}, {<other3>\_1.carry = newnode(<operator\_lowPrecedence>.symbol, <other3>\_1.inh, <arithmeticTerm>.carry)}, {free(<other3>\_2)}, {free(<operator\_lowPrecedence>)}, {free(<arithmeticTerm>)}

1. <other3> ===> \_epsilon\_

{<other3>.carry = <other3>.inh}, {free(\_epsilon\_)}

1. <arithmeticTerm> ===> <factor> <other4>

{<other4>.inh = <factor>.carry}, {<arithmeticTerm>.carry = <other4>.carry}, {free(<other4>)}

1. <other4> ===> <operator\_highPrecedence> <factor> <other4>

{<other4>\_2.inh = <factor>.carry}, {<other4>\_1.carry = newnode(<operator\_highPrecedence>.symbol, <other4>\_1.inh, <factor>.carry)}, {free(<other4>\_2)}, {free(<operator\_highPrecedence>)}, {free(<factor>)}

1. <other4> ===> \_epsilon\_

{<other4>.carry = <other4>.inh}, {free(\_epsilon\_)}

1. <factor> ===> OP <arithmeticExpression> CL

{free(OP)}, {free(CL)}, {<factor>.carry = <arithmeticExpression>.carry}, {free(<arithmeticExpression>)}

1. <factor> ===> <var>

{<factor>.carry = <var>.carry}, {free(<var>)}

1. <operator\_lowPrecedence> ===> PLUS

{<operator\_lowPrecedence>.carry = PLUS.addr}

1. <operator\_lowPrecedence> ===> MINUS

{<operator\_lowPrecedence>.carry = MINUS.addr}

1. <operator\_highPrecedence> ===> MUL

{<operator\_highPrecedence>.carry = MUL.addr}

1. <operator\_highPrecedence> ===> DIV

{<operator\_highPrecedence>.carry = DIV.addr}

1. <booleanExpression> ===> OP <booleanExpression> CL <logicalOp> OP <booleanExpression> CL

{free(OP\_1)}, {free(CL\_1)}, {free(OP\_2)}, {free(CL\_2)}, {<booleanExpression>\_1 = newnode(<logicalOp>.carry.symbol, <booleanExpression>\_2, <booleanExpression>\_3)}

1. <booleanExpression> ===> <constrainedVars> <relationalOp> <constrainedVars>

{<booleanExpression> = newnode(<relationalOp>.carry.symbol, <constrinedVars>\_1.carry, <constrinedVars>\_2.carry)}, {free(<constrainedVars>\_1)}, {free(<constrainedVars>\_2)}

1. <booleanExpression> ===> NOT OP <booleanExpression> CL

{free(NOT)}, {free(OP)}, {free(CL)}, {<booleanExpression>\_1 = newnode("NOT", <booleanExpression>\_2)}

1. <constrainedVars> ===> ID

{<constrinedVars>.carry = ID.addr}

1. <constrainedVars> ===> NUM

{<constrinedVars>.carry = NUM.addr}

1. <constrainedVars> ===> RNUM

{<constrinedVars>.carry = RNUM.addr}

1. <var> ===> ID <matrixElementExtension>

{ID.link = <matrixElementExtension>.carry}, {<var>.carry = ID}, {free(<matrixElementExtension>)}

1. <var> ===> NUM

{<var>.carry = NUM.addr}

1. <var> ===> RNUM

{<var>.carry = RNUM.addr}

1. <var> ===> STR

{<var>.carry = STR.addr}

1. <var> ===> <matrix>

{<var>.carry = <matrix>.addr}

1. <matrix> ===> SQO <rows> SQC

{free(SQO)}, {free(SQC)}, {<matrix> = newnode("MATRIX", <rows>.carry)}, {free(<rows>)}

1. <rows> ===> <row> <other5>

{<row>.link = <other5>.carry}, {<rows>.carry = <row>.addr}, {free(<other5>)}

1. <other5> ===> SEMICOLON <row> <other5>

{<row>.link = <other5>\_2.carry}, {free(<other5>\_2)}, {free(SEMICOLON)}, {<other5>\_1.carry = <row>.addr}

1. <other5> ===> \_epsilon\_

{<other5>.carry = NULL}, {free(\_epsilon\_)}

1. <row> ===> NUM <remainingColElements>

{NUM.link = <remainingColElements>.carry}, {free(<remainingColElements>)}

1. <remainingColElements> ===> COMMA NUM <remainingColElements>

{free(COMMA)}, {<remainingColElements>\_1.carry = NUM.addr}, {NUM.link = <remainingColElements>\_2.carry}, {free(<remainingColElements>\_2)}

1. <remainingColElements> ===> \_epsilon\_

{<remainingColElements>.carry = NULL}, {free(\_epsilon\_)}

1. <matrixElementExtension> ===> SQO NUM COMMA NUM SQC

{free(SQO)}, {free(COMMA)}, {free(SQC)}, {<matrixElemenetExtension>.carry = newnode("indices", NUM\_1.addr, NUM\_2.addr)}

1. <matrixElementExtension> ===> \_epsilon\_

{<matrixElementExtension>.carry = NULL}, {free(\_epsilon\_)}

1. <logicalOp> ===> AND

{<logicalOp>.carry = AND.addr}

1. <logicalOp> ===> OR

{<logicalOp>.carry = OR.addr}

1. <relationalOp> ===> LT

{<relationalOp>.carry = LT.addr}

1. <relationalOp> ===> LE

{<relationalOp>.carry = LE.addr}

1. <relationalOp> ===> EQ

{<relationalOp>.carry = EQ.addr}

1. <relationalOp> ===> GT

{<relationalOp>.carry = GT.addr}

1. <relationalOp> ===> GE

{<relationalOp>.carry = GE.addr}

1. <relationalOp> ===> NE

{<relationalOp>.carry = NE.addr}