**Semantic Rules for AST Creation**

1. <program> --> <otherFunctions> <mainFunction>
   * <program>.treenode = createChildren(<otherFunctions>.treenode, <mainFunction>.treenode);
2. <mainFunction> --> TK\_MAIN <stmts> TK\_END
   * <mainFunction>.treenode = createTreeNode(‘main’, <stmts>.treenode);
3. <otherFunctions> --> <function> <otherFunctions>1
   * <otherFunctions>.treenode = createTreeNodeList(head = <function>.treenode, tail = <otherFunctions>1.treenode);

1. <otherFunctions> --> EPS
   * <otherFunctions>.treenode = NULL;
2. <function> --> TK\_FUNID <input\_par> <output\_par> TK\_SEM <stmts> TK\_END
   * <function>.treenode = createTreeNode(TK\_FUNID.value, <input\_par>.treenode, <output\_par>.treenode, <stmts>.treenode);
3. <input\_par> --> TK\_INPUT TK\_PARAMETER TK\_LIST TK\_SQL <parameter\_list> TK\_SQR
   * <input\_par>.treenode = <parameter\_list>.treenode;
4. <output\_par> --> TK\_OUTPUT TK\_PARAMETER TK\_LIST TK\_SQL <parameter\_list> TK\_SQR
   * <output\_par>.treenode = <parameter\_list>.treenode;
5. <output\_par> --> EPS
   * <output\_par>.treenode = NULL;
6. <parameter\_list> --> <dataType> TK\_ID <remaining\_list>
   * newnode = createTreeNode(<dataType>.name, TK\_ID.value);
   * <parameter\_list>.treenode = createTreeNodeList(head = newnode, tail = <remainingList>.treenode);
7. <dataType> --> <primitiveDatatype>
   * <dataType>.name = <primitiveDatatype>.name;
8. <dataType> --> <constructedDatatype>
   * <dataType>.name = <constructedDatatype>.name;
9. <primitiveDatatype> --> TK\_INT
   * <primitiveDatatype>.name = ‘int’;
10. <primitiveDatatype> --> TK\_REAL
    * <primitiveDatatype>.name= ‘real’;
11. <constructedDatatype> --> TK\_RECORD TK\_RECORDID
    * <constructedDatatype>.name = TK\_RECORDID.name;
12. <remaining\_list> --> TK\_COMMA <parameter\_list>
    * <remaining\_list>.treenode = <parameter\_list>.treenode;
13. <remaining\_list> --> EPS
    * <remaining\_list>.treenode = NULL;
14. <stmts> --> <typeDefinitions> <declarations> <otherStmts> <returnStmt>
    * <stmts>.treenode = createTreeNode(<typeDefinitions>.treenode, <declarations>.treenode, <otherStmts>.treenode, <returnStmt>.treenode);
15. <typeDefinitions> --> <typeDefinition> <typeDefinitions>1
    * <typeDefinitions>.treenode = createTreeNodeList(head = <typeDefinition>.treenode, tail = <typeDefinitions>1..treenode);
16. <typeDefinitions> --> EPS
    * <typeDefinitions>.treenode = NULL;
17. <typeDefinition> --> TK\_RECORD TK\_RECORDID <fieldDefinitions> TK\_ENDRECORD TK\_SEM
    * <typeDefinition>.treenode = createTreeNode(TK\_RECORDID., <fieldDefinitions>.treenode);
18. <fieldDefinitions> --> <fieldDefinition>1 <fieldDefinition>2 <moreFields>
    * newnode = createTreeNodeList(head = <fieldDefinition>2 .treenode, tail = <moreFields>.treenode);
    * <fieldDefinitions>.treenode = createTreeNodeList(head = <fieldDefinition>1.treenode, tail = newnode);
19. <fieldDefinition> --> TK\_TYPE <primitiveDatatype> TK\_COLON TK\_FIELDID TK\_SEM
    * <fieldDefinition>.treenode = createTreeNode(<primitiveDatatype>.name , TK\_FIELDID.name);
20. <moreFields> --> <fieldDefinition> <moreFields>1
    * <moreFields>.treenode = createTreeNodeList(head = <fieldDefinition>.treenode, tail = <moreFields>1.treenode);
21. <moreFields> --> EPS
    * <moreFields>.treenode = NULL;
22. <declarations> --> <declaration> <declarations>1
    * <declarations>.treenode = createTreeNodeList(head = <declaration>.treenode, tail = <declarations>1.treenode);
23. <declarations> --> EPS
    * <declarations>.treenode = NULL;
24. <declaration> --> TK\_TYPE <dataType> TK\_COLON TK\_ID <global\_or\_not> TK\_SEM
    * <declaration>.treenode = createTreeNode(<dataType>.name, TK\_ID.value, <global\_or\_not>.global);
25. <global\_or\_not> --> TK\_COLON TK\_GLOBAL
    * <global\_or\_not>.global = True;
26. <global\_or\_not> --> EPS
    * <global\_or\_not>.global = False;
27. <otherStmts> --> <stmt> <otherStmts>
    * <otherStmts>.treenode = createTreeNode(<stmt>.treenode, <otherStmts>.treenode);
28. <otherStmts> --> EPS
    * <otherStmts>.treenode = NULL;
29. <stmt> --> <assignmentStmt>
    * <stmt>.treenode = <assignmentStmt>.treenode;
30. <stmt> --> <iterativeStmt>
    * <stmt>.treenode = <iterativeStmt>.treenode;
31. <stmt> --> <conditionalStmt>
    * <stmt>.treenode = <conditionalStmt>.treenode;
32. <stmt> --> <ioStmt>
    * <stmt>.treenode = <ioStmt>.treenode;
33. <stmt> --> <funCallStmt>
    * <stmt>.treenode = <funCallStmt>.treenode;
34. <assignmentStmt> --> <SingleOrRecId> TK\_ASSIGNOP <arithmeticExpression> TK\_SEM
    * <assignmentStmt>.treenode = createTreeNode(‘<---’, <SingleOrRecId>.treenode, <arithmeticExpression>.treenode);
35. <SingleOrRecId> --> TK\_ID <new\_24>
    * <SingleOrRecId>.treenode = createTreeNode(TK\_ID.value, fieldVal = <new\_24>.treenode);

1. <new\_24> --> TK\_DOT TK\_FIELDID
   * <new\_24>.treenode = TK\_FIELDID.treenode;
2. <new\_24> --> EPS
   * <new\_24>.treenode = NULL;
3. <funCallStmt> --> <outputParameters> TK\_CALL TK\_FUNID TK\_WITH TK\_PARAMETERS <inputParameters> TK\_SEM
   * <funCallStmt>.treenode = createTreeNode(TK\_FUNID.value, <outputParameters>.treenode, <inputParameters>.treenode);
4. <outputParameters> --> TK\_SQL <idList> TK\_SQR TK\_ASSIGNOP
   * <outputParameters>.treenode = createTreeNode(‘<---’, <idList>.treenode);
5. <outputParameters> --> EPS
   * <outputParameters>.treenode = NULL;
6. <inputParameters> --> TK\_SQL <idList> TK\_SQR
   * <inputParameters>.treenode = <idList>.treenode;
7. <iterativeStmt> --> TK\_WHILE TK\_OP <booleanExpression> TK\_CL <stmt> <otherStmts> TK\_ENDWHILE
   * newnode = createTreeNodeList(head = <stmt>.treenode, tail = <otherStmts>.treenode);
   * <iterativeStmt>.treenode = createTreeNode(‘while’, <booleanExpression>.treenode, newnode);
8. <conditionalStmt> --> TK\_IF TK\_OP <booleanExpression> TK\_CL TK\_THEN <stmt> <otherStmts> <elsePart>
   * newnode = createTreeNodeList(head = <stmt>.treenode, tail = <otherStmts>.treenode);
   * <conditionalStmt> = createTreeNode(‘if’, <booleanExpression>.treenode, newnode, <elsePart>.treenode);
9. <elsePart> --> TK\_ELSE <stmt> <otherStmts> TK\_ENDIF
   * newnode = createTreeNodeList(head = <stmt>.treenode, tail = <otherStmts>.treenode);
   * <elsePart>.treenode = createTreeNode(‘else’, newnode);
10. <elsePart> --> TK\_ENDIF
    * <elsePart>.treenode = NULL;
11. <ioStmt> --> TK\_READ TK\_OP <SingleOrRecId> TK\_CL TK\_SEM
    * <ioStmt>.treenode = createTreeNode(‘read’, <SingleOrRecId>.treenode);
12. <ioStmt> --> TK\_WRITE TK\_OP <all> TK\_CL TK\_SEM
    * <ioStmt>.treenode = createTreeNode(‘write’, <all>.treenode);
13. <arithmeticExpression> --> <term> <expPrime>
    * <arithmeticExpression>.treenode = <expPrime>.syn;
    * <expPrime>.inh = <term>.treenode;
14. <expPrime> --> <lowPrecedenceOperators> <term> <expPrime>1
    * <expPrime>1.inh = createTreeNode(<lowPrecedenceOperators>.name, <expPrime>.inh, <term>.treenode);
    * <expPrime>.syn = <expPrime>1.syn;
15. <expPrime> --> EPS
    * <expPrime>.syn = <expPrime>.inh;
16. <term> --> <factor> <termPrime>
    * <term>.treenode = <termPrime>.syn;
    * <termPrime>.inh = <factor>.treenode;
17. <termPrime> --> <highPrecedenceOperators> <factor> <termPrime>1
    * <termPrime>1.treenode = createTreeNode(<highPrecedenceOperators>.name, <termPrime>.inh, <factor>.treenode);
    * <termPrime>.syn = <termPrime>1.syn;
18. <termPrime> --> EPS
    * <termPrime>.syn = <termPrime>.inh;
19. <factor> --> TK\_OP <arithmeticExpression> TK\_CL
    * <factor>.treenode = <arithmeticExpression>.treenode;
20. <factor> --> <all>
    * <factor>.treenode = <all>.treenode;
21. <highPrecedenceOperators> --> TK\_MUL
    * <highPrecedenceOperators>.name = ‘\*’;
22. <highPrecedenceOperators> --> TK\_DIV
    * <highPrecedenceOperators>.name = ‘/’;
23. <lowPrecedenceOperators> --> TK\_PLUS
    * <lowPrecedenceOperators>.name = ‘+’;
24. <lowPrecedenceOperators> --> TK\_MINUS
    * <lowPrecedenceOperators>.name = ‘-’;
25. <all> --> TK\_NUM
    * <all>.treenode = createTreeNode(TK\_NUM.value);
26. <all> --> TK\_RNUM
    * <all>.treenode = createTreeNode(TK\_RNUM.value);
27. <all> --> TK\_ID <temp>
    * <all>.treenode = createTreeNode(TK\_ID.value, <temp>.treenode);
28. <temp> --> EPS
    * <temp>.treenode = NULL;
29. <temp> --> TK\_DOT TK\_FIELDID
    * <temp>.treenode = TK\_FIELDID.treenode;
30. <booleanExpression> --> TK\_OP <booleanExpression>1 TK\_CL <logicalOp> TK\_OP <booleanExpression>2 TK\_CL
    * <booleanExpression>.treenode = createTreeNode(<logicalOp>.name, <booleanExpression>1.treenode, <booleanExpression>2.treenode);

1. <booleanExpression> --> <var>1 <relationalOp> <var>2
   * <booleanExpression>.treenode = createTreeNode(<relationalOp>.treenode, <var>1.treenode, <var>2.treenode);
2. <booleanExpression> --> TK\_NOT TK\_OP <booleanExpression>1 TK\_CL
   * <booleanExpression>.treenode = createTreeNode(‘~’, <booleanExpression>1.treenode);
3. <var> --> TK\_ID
   * <var>.treenode = createTreeNode(TK\_ID.value);
4. <var> --> TK\_NUM
   * <var>.treenode = createTreeNode(TK\_NUM.value);
5. <var> --> TK\_RNUM
   * <var>.treenode = createTreeNode(TK\_RNUM.value);
6. <logicalOp> --> TK\_AND
   * <logicalOp>.name = ‘&&&’;
7. <logicalOp> --> TK\_OR
   * <logicalOp>.name = ‘@@@’;
8. <relationalOp> --> TK\_LT
   * <relationalOp>.name = ‘<’;
9. <relationalOp> --> TK\_LE
   * <relationalOp>.name = ‘<=’;
10. <relationalOp> --> TK\_EQ
    * <relationalOp>.name = ‘==’;
11. <relationalOp> --> TK\_GT
    * <relationalOp>.name = ‘>’;
12. <relationalOp> --> TK\_GE
    * <relationalOp>.name = ‘>=’;
13. <relationalOp> --> TK\_NE
    * <relationalOp>.name = ‘!=’;
14. <returnStmt> --> TK\_RETURN <optionalReturn> TK\_SEM
    * <returnStmt>.treenode = <optionalReturn>.treenode;
15. <optionalReturn> --> TK\_SQL <idList> TK\_SQR
    * <optionalReturn>.treenode = <idList>.treenode;

1. <optionalReturn> --> EPS
   * <optionalReturn>.treenode = NULL;
2. <idList> --> TK\_ID <more\_ids>
   * <idList>.treenode = createTreeNodeList(head = TK\_ID.value, tail = <more\_ids>.treenode);

1. <more\_ids> --> TK\_COMMA <idList>
   * <more\_ids>.treenode = <idList>.treenode;
2. <more\_ids> --> EPS
   * <more\_ids>.treenode = NULL;