Rules for AST Generation

Production	Semantic Rule

<program> ===> <otherFunctions> <mainFunction>	Program.node = new Node(program, otherFunctions.node)
	if(otherFunctions.node != NULL)
	  otherFunctions.node.sibling = mainFunction.node
	Else
	  program.node.firstChild = mainFunction.node

<mainFunction>===> TK_MAIN <stmts> TK_END	Main.node = new node (main, stmts.node)
<otherFunctions>===> <function><otherFunctions1>	otherFunctions.node = function.node Function.node.sibling = otherFunctions1.node
<otherFunctions>===> eps	otherFunctions.node = NULL
<function>===>TK_FUNID <input_par> <output_par> TK_SEM <stmts> TK_END	function.node=new Node(function, new Leaf(TK_FUNID)) function.node.firstChild.sibling = input_par.node Input_par.node.sibling = output_par.node Output_par.node.sibling = stmts.node
<input_par>===>TK_INPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR	input_par.node=new node(input_par,parameter_list.node)
<output_par>===>TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR	output_par.node=new node(output_par, parameter_list.node)
<output_par>===>eps	Output_par.node = new node(output_par,NULL)
<parameter_list>===><dataType> TK_ID <remaining_list>	dataType.node.sibling = new Leaf(TK_ID) dataType.node.sibling.sibling= remaining_list.node parameter_list.node= dataType.node
<dataType>===> <primitiveDatatype>	dataType.node=primitiveDatatype.node
<dataType>===> <constructedDatatype>	dataType.node=constructedDatatype.node
<primitiveDatatype>===> TK_REAL	primitiveDatatype.node= new Leaf(TK_REAL)
<primitiveDatatype>===> TK_INT	primitiveDatatype.node= new Leaf(TK_INT)
<constructedDatatype>===>TK_RECORD TK_RECORDID	constructedDatatype.node=new Leaf(TK_RECORDID)
<remaining_list>===>TK_COMMA <parameter_list>	remaining_list.node=parameter_list.node
<remaining_list>===>eps	remaining_list.node=NULL
<stmts>===><typeDefinitions> <declarations> <otherStmts><returnStmt>	Os = new node (otherStmts, otherStmts.node) Os.sibling = returntStmt.node d = new node(declarations, declarations.node) D.sibling = os Td = new node(typeDefinitions, typeDefinitions.node) Td.sibling = D stmts.node= new Node(stmt,Td.node)

<typeDefinitions>===><typeDefinition><typeDefinitions1>	typeDefinitions.node = typeDefinition.node typeDefinition.node.sibling = typeDefinitions1.node
<typeDefinitions>===>eps	typeDefinitions.node=NULL
<typeDefinition>===>TK_RECORD TK_RECORDID <fieldDefinitions> TK_ENDRECORD TK_SEM	Rid = new leaf(TK_RECORDID, token) Rid.sibling = fieldDefinitions.node typeDefinition.node=new Node(typeDefinition, Rid)
<fieldDefinitions>===> <fieldDefinition1> <fieldDefinition2> <moreFields>	fieldDefinitions=new Node(fieldDefinitions, fieldDefintion1.node) FD1.node.sibling.sibling = FD2.node FD2.node.sibling.sibling = moreFields.node
<fieldDefinition>===> TK_TYPE <primitiveDatatype> TK_COLON TK_FIELDID TK_SEM	Fid = new Leaf(TK_FIELDID, token) primitiveDataType.node.sibling = Fid. fieldDefinition.node= primitiveDatatype.node
<moreFields>===><fieldDefinition><moreFields1>	moreFields.node = fieldDef.node fieldDefinition.node.sibling.sibling = moreFields1.node
<moreFields>===>eps	moreFields.node=NULL
<declarations> ===> <declaration><declarations1>	Declarations.node = declaration.node Declaration.node.sibling = declarations1.node
<declarations> ===> eps	declarations.node= NULL
<declaration>===> TK_TYPE <dataType> TK_COLON TK_ID <global_or_not> TK_SEM	Id =new Leaf(TK_TID, token) Id.sibling = global_or_not.node dataType.node.sibling = id declaration.node= new Node (declaration, dataType.node)
<global_or_not>===>TK_COLON TK_GLOBAL	globalNot.node= new Leaf (TK_GLOBAL, token)
<global_or_not>===> eps	globalNot.node= NULL
<otherStmts>===> <stmt><otherStmts1>	Stmt.node.sibling = otherStmts1.node otherStmts.node= stmt.node
<otherStmts>===> eps	otherStms.node=NULL
<stmt>===> <assignmentStmt>	stmt.node=assignmentStmt.node
<stmt>===> <iterativeStmt>	stmt.node=iterativeStmt.node
<stmt>===> <conditionalStmt>	stmt.node=conditionalStmt.node
<stmt>===> <ioStmt>	stmt.node=ioStmt.node
<stmt>===> <funCallStmt>	stmt.node=funCallStmt.node
<assignmentStmt>===><singleOrRecId> TK_ASSIGNOP <arithmeticExpression> TK_SEM	singleOrRecid.node.sibling=arithmeticExpression.node assignmentStmt.node= new Node(assignmentStmt , singleOrRecId.node)

<singleOrRecId>===>TK_ID <new_24>	Id = new leaf(TK_ID, token) id.sibling= new_24.node singleOrRecId.node=new Node (singleOrRecid , id)
<new_24> ===> TK_DOT TK_FIELDID	new_24.node=new Leaf(TK_FIELDID, token)
<new_24> ===> eps	new_24.node=NULL
<funCallStmt>===><outputParameters> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS <inputParameters> TK_SEM	outputParameters.node.sibling = inputPrameters.node Fid = new leaf(TK_FUNID, token) Fid.sibling = outputParameters.node funCallStmt.node = new Node(funCallStmt, fid)
<outputParameters> ==> TK_SQL <idList> TK_SQR TK_ASSIGNOP	outputParameters.node= new Node(outputParameters,idList.node)
<outputParameters> ==> eps	outputParameters.node=NULL
<inputParameters>===> TK_SQL <idList> TK_SQR	inputParameters.node= new Node(inputParameters,idList.node)
<iterativeStmt>===> TK_WHILE TK_OP <booleanExpression> TK_CL <stmt><otherStmts> TK_ENDWHILE	Stmt.node.sibling = otherStmts.node Os = new node(otherStmts, stmt.node) booleanExpression.node.sibling = os.node iterativeStmt.node=new Node(iterativeStmt, booleanExpression.node)
<conditionalStmt>===> TK_IF TK_OP <booleanExpression> TK_CL TK_THEN <stmt><otherStmts> <elsePart>	Stmt.node.sibling = otherStmts.node if = new node(TK_IF, stmt.node) if.sibling= elsePart.node booleanExpression.node.sibling = if conditionalStmt.node=new Node(conditionalStmt, booleanExpression.node)
<elsePart>===>TK_ELSE <stmt><otherStmts> TK_ENDIF	Stmt.node.sibling = otherStmts.node elsePart.node = new Node(TK_ELSE, stmt.node)
<elsePart>===>TK_ENDIF	elsePart.node = new Node(TK_ELSE, NULL)
<ioStmt>===> TK_READ TK_OP <singleOrRecId> TK_CL TK_SEM	ioStmt.node=new Node(read,singleOrRecId.node)
<ioStmt>===> TK_WRITE TK_OP <allVar> TK_CL TK_SEM	iostmt.node= new Node(write, allVar.node)
<allVar>===>TK_NUM	allVar.node= new Leaf(TK_NUM, tokenl)
<allVar>===>TK_RNUM	allVar.node= new Leaf(TK_RNUM, token)
<allVar>===>TK_ID<allVarTemp>	Id = new leaf(TK_ID, token) id.sibling= allVarTemp.node allVar.node=new Node (allVar , id)
<allVarTemp>===>TK_DOT TK_FIELDID	allVarTemp.node= new leaf (TK_FIELDID , token)
<allVarTemp>===>eps	allVarTemp.node = NULL

<arithmeticExpression> ===> <term> <expPrime>	expPrime.inh = term.node arithmeticExpression.node = new node(arithmeticExpression, expPrime.node)
<expPrime> ===> <lowPrecedenceOperators> <term> <expPrime$_1$>	lowPrecedenceOp.node.firstChild = expPrime.inh expPrime1.inh = term.node lowPrecedenceOp.node.firstChild.sibling = expPrime1.node expPrime.node = lowPrecedenceOp.node
<expPrime> ===> eps	expPrime.node=expPrime.inh
<lowPrecedenceOperators>===> TK_PLUS	lowPrecedenceOp.node= new node(TK_PLUS, NULL)
<lowPrecedenceOperators>===> TK_MINUS	lowPrecedenceOp.node= new node(TK_MINUS, NULL)
<term>===> <factor> <termPrime>	termPrime.inh = factor.node term.node = termPrime.node
<termPrime> ===> <highPrecedenceOperators><factor> <termPrime$_1$>	highPrecedenceOp.node.firstChild = termPrime.inh termPrime1.inh = factor.node highPrecedenceOp.node.firstChild.sibling = termPrime1.node termPrime.node= highPrecedenceOp.node
<termPrime> ===> eps	termPrime.node = termPrime.inh
<highPrecedenceOperators>===> TK_DIV	highPrecedenceOperators.node=new Leaf (TK_DIV, NULL)
<highPrecedenceOperators>===> TK_MUL	highPrecedenceOperators.node=new Leaf (TK_MUL, NULL)
<factor> ===> <all>	factor.node=all.node
<factor> ===> TK_OP <arithmeticExpression> TK_CL	factor.node=arithmeticExpression.node
<all>===> TK_ID<temp>	Id = new leaf(TK_ID, token) id.sibling= Temp.node all.node=new Node(all , id)
<all>===>TK_NUM	All.node = new Leaf(TK_NUM, tokenl)
<all>===> TK_RNUM	all.node= new Leaf(TK_NUM, tokenl)
<temp>===> TK_DOT TK_FIELDID	Temp.node = new leaf(TK_FIELDID, token)
<temp>===> eps	Temp.node = NULL
<booleanExpression>===>TK_OP <booleanExpression$_1$> TK_CL <logicalOp> TK_OP <booleanExpression$_2$> TK_CL	booleanExpression1.node.sibling = booleanExpression2.node logicalOp.inh = booleanExpression1.node booleanExpression.node = logicalOp.node
<booleanExpression>===><var$_1$> <relationalOp> <var$_2$>	Var1.node.sibling = var2.node relationalOp.inh = var1.node booleanExpression.node=relationalOp.node

<booleanExpression>===>TK_NOT TK_OP <booleanExpression$_1$>TK_CL	booleanExpression.node = new node(TK_NOT,booleanExpression$_1$,node)
<var>===> TK_ID	var.node= new Leaf (TK_ID,token)
<var>===> TK_NUM	var.node= new Leaf (TK_NUM,token)
<var>===> TK_RNUM	var.node= new Leaf (TK_RNUM,token)
<logicalOp>===>TK_AND	logicalOp.node= new Node(TK_AND,logicalOp.inh)
<logicalOp>===>TK_OR	logicalOp.node= new Leaf(TK_OR,logicalOp.inh)
<relationalOp>===> TK_LT | TK_LE | TK_EQ |TK_GT | TK_GE | TK_NE	relationalOp.node = new Node (TK_LT,relationalOp.inh) etc..
<returnStmt>===>TK_RETURN <optionalReturn> TK_SEM	returnStmt.node = new node(returnStmt, optionalReturn.node)
<optionalReturn>===>TK_SQL <idList> TK_SQR	optionalReturn.node = idList.node
<optionalReturn>===>eps	optRet.node=NULL
<idList>===> TK_ID <more_ids>	Temp = new leaf(TK_ID, token) Temp.sibling = more_ids.node idList.node = temp
<more_ids>===> TK_COMMA <idList>	more_ids.node=idList.node
<more_ids>===> eps	More_ids.node = NULL

Grammar for Reference

1. <program> ===> <otherFunctions> <mainFunction>
2. <mainFunction>===> TK_MAIN <stmts> TK_END
3. <otherFunctions>===> <function><otherFunctions> | eps
4. <function>===>TK_FUNID <input_par> <output_par> TK_SEM <stmts> TK_END
5. <input_par>===>TK_INPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR
6. <output_par>===>TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR |eps
7. <parameter_list>===><dataType> TK_ID <remaining_list>
8. <dataType>===> <primitiveDatatype> |<constructedDatatype>
9. <primitiveDatatype>===> TK_INT | TK_REAL
10. <constructedDatatype>===>TK_RECORD TK_RECORDID
11. <remaining_list>===>TK_COMMA <parameter_list> | eps
12. <stmts>===><typeDefinitions> <declarations> <otherStmts><returnStmt>
13. <typeDefinitions>===><typeDefinition><typeDefinitions> |eps
14. <typeDefinition>===>TK_RECORD TK_RECORDID <fieldDefinitions> TK_ENDRECORD TK_SEM
15. <fieldDefinitions>===> <fieldDefinition><fieldDefinition><moreFields>
16. <fieldDefinition>===> TK_TYPE <primitiveDatatype> TK_COLON TK_FIELDID TK_SEM
17. <moreFields>===><fieldDefinition><moreFields> | eps
18. <declarations> ===> <declaration><declarations>|eps
19. <declaration>===> TK_TYPE <dataType> TK_COLON TK_ID <global_or_not> TK_SEM
20. <global_or_not>===>TK_COLON TK_GLOBAL| eps
21. <otherStmts>===> <stmt><otherStmts> | eps
22. <stmt>===> <assignmentStmt> | <iterativeStmt>|<conditionalStmt>|<ioStmt>| <funCallStmt>
23. <assignmentStmt>===><singleOrRecId> TK_ASSIGNOP <arithmeticExpression> TK_SEM
24. <singleOrRecId>===>TK_ID <new_24>
25. <new_24> ===> eps | TK_DOT TK_FIELDID
26. <funCallStmt>===><outputParameters> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS <inputParameters> TK_SEM
27. <outputParameters> ==> TK_SQL <idList> TK_SQR TK_ASSIGNOP | eps
28. <inputParameters>===> TK_SQL <idList> TK_SQR
29. <iterativeStmt>===> TK_WHILE TK_OP <booleanExpression> TK_CL <stmt><otherStmts> TK_ENDWHILE
30. <conditionalStmt>===> TK_IF TK_OP <booleanExpression> TK_CL TK_THEN <stmt><otherStmts> <elsePart>
31. <elsePart>===>TK_ELSE <stmt><otherStmts> TK_ENDIF | TK_ENDIF
32. <ioStmt>===> TK_READ TK_OP <singleOrRecId> TK_CL TK_SEM| TK_WRITE TK_OP <allVar> TK_CL TK_SEM
33. <allVar>===>TK_NUM|TK_RNUM|TK_ID<allVarTemp>
34. <allVarTemp>===>TK_DOT TK_FIELDID|eps

35. <arithmeticExpression> ===> <term> <expPrime>
36. <expPrime> ===> <lowPrecedenceOperators> <term> <expPrime> | eps
37. <term>===> <factor> <termPrime>
38. <termPrime> ===> <highPrecedenceOperators><factor> <termPrime> | eps
39. <factor> ===> TK_OP <arithmeticExpression> TK_CL | <all>
40. <highPrecedenceOperators>===> TK_MUL | TK_DIV
41. <lowPrecedenceOperators> ===> TK_PLUS | TK_MINUS
42. <all>===> TK_ID<temp>|TK_NUM|TK_RNUM
43. <temp>===> eps | TK_DOT TK_FIELDID
44. <booleanExpression>===>TK_OP <booleanExpression> TK_CL <logicalOp> TK_OP <booleanExpression> TK_CL| <var> <relationalOp> <var>|TK_NOT TK_OP <booleanExpression>TK_CL
45. <var>===> TK_ID|TK_NUM|TK_RNUM
46. <logicalOp>===>TK_AND | TK_OR
47. <relationalOp>===> TK_LT | TK_LE | TK_EQ |TK_GT | TK_GE | TK_NE
48. <returnStmt>===>TK_RETURN <optionalReturn> TK_SEM
49. <optionalReturn>===>TK_SQL <idList> TK_SQR | eps
50. <idList>===> TK_ID <more_ids>
51. <more_ids>===> TK_COMMA <idList> | eps