AST SEMANTIC RULES

BATCH No. - 66
Upadhyay Gaurav Arvind – 2013A7PS030P
Shivin Srivastava – 2013A7PS073P

```
<program> ===> <otherFunctions> <mainFunction>
{<program>.ptr = newNode([<otherFunctions>.ptr, <mainFunction>.ptr])}

<mainFunction> ===> TK_MAIN <stmts> TK_END
{<mainFunction>.ptr = newNode([makeLeaf("TK_MAIN", "main"), <stmts>.ptr])}

<otherFunctions> ===> <function> <otherFunctions>
{<otherFunctions>.ptr = newNode([<function>.ptr, <otherFunctions>.ptr])}

<otherFunctions> ===> TK_EPS
{<otherFunctions>.ptr = NULL}

<function> ===> TK_FUNID <input_par> <output_par> TK_SEM <stmts> TK_END
{<function>.ptr = newNode([makeLeaf("TK_FUNID", funId.entry), <input_par>.ptr,
<output_par>.ptr])}

<input_par> ===> TK_INPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR
{<input_par>.ptr = newNode([makeLeaf("TK_INPUT", "input"),
<parameter_list>.ptr])}

<output_par> ===> TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL <parameter_list> TK_SQR
{<output_par>.ptr = newNode([<makeLeaf("TK_OUTPUT", "output")>,
<parameter_list>.ptr])}

<output_par> ===> TK_EPS
[<output_par>.ptr = NULL]

<parameter_list> ===> <dataType> TK_ID <remaining_list>
{<parameter_list>.ptr = newNode([makeLeaf(TK_ID, id.entry), <dataType>.ptr,
<remaining_list>.ptr])}

<dataType> ===> <primitiveDatatype>
{<dataType>.ptr = <constructedDatatype>.ptr}

<dataType> ===> <constructedDatatype>
{<dataType>.ptr = <constructedDatatype>.ptr}

<primitiveDatatype> ===> TK_INT
{<primitiveDatatype>.ptr = makeLeaf("TK_INT", "int")}

<primitiveDatatype> ===> TK_REAL
{<primitiveDatatype>.ptr = makeLeaf("TK_REAL", "real")}

<constructedDatatype> ===> TK_RECORD TK_RECORDID
{<constructedDatatype>.ptr = makeLeaf("TK_RECORD", "recordId.val")}

<remaining_list> ===> TK_COMMA <parameter_list>
{<remaining_list>.ptr=<parameter_list>.ptr}

<remaining_list> ===> TK_EPS

<stmts> ===> <typeDefinitions> <declarations> <otherStmts> <returnStmt>
{<stmts>.ptr = newNode([<typeDefinitions>.ptr, <declarations>.ptr,
<otherStmts>.ptr, <returnStmt>.ptr])}
```

```
<typeDefinitions> ===> <typeDefinition> <typeDefinitions>
{<typeDefinitions>.ptr = newNode([<typeDefinition>.ptr, <typeDefinitions>.ptr])}

<typeDefinitions> ===> TK_EPS
{<typeDefinitions>.ptr = NULL)}

<typeDefinition> ===> TK_RECORD TK_RECORDID <fieldDefinitions> TK_ENDRECORD
TK_SEM
{<typeDefinition>.ptr = newNode([makeLeaf("TK_RECORDID", recordId.entry),
<fieldDefinition>.ptr])}

<fieldDefinitions> ===> <fieldDefinition> <fieldDefinition> <moreFields>
{<fieldDefinitions>.ptr = newNode([<fieldDefinition>.ptr, <fieldDefinition>.ptr,
<moreFields>.ptr])}

<fieldDefinition> ===> TK_TYPE <primitiveDatatype> TK_COLON TK_FIELDID TK_SEM
{<fieldDefinition>.ptr = newNode([makeLeaf("TK_FIELDID", fieldId.entry),
<primitiveDatatype>.ptr])}

<moreFields> ===> <fieldDefinition> <moreFields>
{<moreFields>.ptr = newNode([<fieldDefinition>.ptr, <moreFields>.ptr])}

<moreFields> ===> TK_EPS
{<moreFields>.ptr = NULL)}

<declarations> ===> <declaration> <declarations>
{<declarations>.ptr = newNode([<declaration>.ptr, <declarations>.ptr])}

<declarations> ===> TK_EPS
{<declarations>.ptr = NULL}

<declaration> ===> TK_TYPE <dataType> TK_COLON TK_ID <global_or_not> TK_SEM
{<declaration>.ptr = newNode([makeLeaf("TK_ID", id.entry), <dataType>.ptr,
<global_or_not>.ptr])}

<global_or_not> ===> TK_COLON TK_GLOBAL
{<global_or_not>.ptr = makeLeaf("TK_GLOBAL", "global")}

<global_or_not> ===> TK_EPS
{<global_or_not>.ptr = NULL}

<otherStmts> ===> <stmt> <otherStmts>
{<otherStmts>.ptr = newNode([<stmt>.ptr, <otherStmts>.ptr])}

<otherStmts> ===> TK_EPS
{<otherStmts>.ptr = NULL}

<stmt> ===> <assignmentStmt>
{<stmt>.ptr = <assignmentStmt>.ptr}

<stmt> ===> <iterativeStmt>
{<stmt>.ptr = <iterativeStmt>.ptr}

<stmt> ===> <conditionalStmt>
{<stmt>.ptr = <conditionalStmt>.ptr}

<stmt> ===> <ioStmt>
{<stmt>.ptr = <ioStmt>.ptr}

<stmt> ===> <funCallStmt>
{<stmt>.ptr = <funCallStmt>.ptr}

<assignmentStmt> ===> <singleOrRecId> TK_ASSIGNOP <arithmeticExpression> TK_SEM
{<assignmentStmt>.ptr = newNode([makeLeaf("TK_ASSIGNOP", "="),
```

```
<singleOrRecId>.ptr,
<arithmeticExpression>.ptr])}

<singleOrRecId> ===> TK_ID <new_24>
{<singleOrRecId>.ptr = newNode([makeLeaf("TK_ID", id.entry), <new_24>.ptr])}

<new_24> ===> TK_DOT TK_FIELDID
{<new_24>.ptr = newNode([makeLeaf("TK_FIELDID", fieldId.entry)])}

<new_24> ===> TK_EPS
{<new_24>.ptr = NULL])}

<funCallStmt> ===> <outputParameters> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS
<inputParameters> TK_SEM
{<funCallStmt>.ptr = newNode([makeLeaf("TK_FUNID", funId.entry),
<outputParameters>.ptr,
<inputParameters>.ptr])}

<outputParameters> ===> TK_SQL <idList> TK_SQR TK_ASSIGNOP
{<outputParameters>.ptr = <idList>.ptr}

<outputParameters> ===> TK_EPS
{<outputParameters>.ptr = NULL}

<inputParameters> ===> TK_SQL <idList> TK_SQR
{<inputParameters>.ptr = <idList>.ptr}

<iterativeStmt> ===> TK_WHILE TK_OP <booleanExpression> TK_CL <stmt>
<otherStmts> TK_ENDWHILE
{<iterativeStmt>.ptr = newNode([makeLeaf("TK_WHILE", "while"),
<booleanExpression>.ptr, <stmt>.ptr, <otherStmts>.ptr])}

<conditionalStmt> ===> TK_IF TK_OP <booleanExpression> TK_CL TK_THEN <stmt>
<otherStmts> <elsePart>
{<conditionalStmt>.ptr = newNode([makeLeaf("TK_IF", "if"),
<booleanExpression>.ptr, <stmt>.ptr, <otherStmts>.ptr, <elsePart>.ptr])}

<elsePart> ===> TK_ELSE <stmt> <otherStmts> TK_ENDIF
{<elsePart>.ptr = make Node([makeLeaf("TK_ELSE", "else"), <stmt>.ptr,
<otherStmts>.ptr])}

<elsePart> ===> TK_ENDIF
#do nothing

<ioStmt> ===> TK_READ TK_OP <singleOrRecId> TK_CL TK_SEM
{<ioStmt>.ptr = make Node([makeLeaf("TK_READ"), <singleOrRecId>.ptr]}

<ioStmt> ===> TK_WRITE TK_OP <allVar> TK_CL TK_SEM
{<ioStmt>.ptr = make Node([makeLeaf("TK_WRITE"), <allVar>.ptr]}

<allVar> ===> <singleOrRecId>
{<allVar>.ptr = <singleOrRecId>.ptr}

<allVar> ===> TK_NUM
{<var>.ptr = makeLeaf("TK_NUM", num.val)}

<allVar> ===> TK_RNUM
{<var>.ptr = makeLeaf("TK_RNUM", rnum.val)}

<arithmeticExpression> ===> <term> <expPrime>
{<arithmeticExpression>.ptr = newNode([<term>.ptr, <expPrime>.ptr])}

<expPrime> ===> <lowPrecedenceOperators> <term> <expPrime>
{<expPrime>.ptr = newNode([<lowPrecedenceOperators>.ptr, <term>.ptr,
```

```
<expPrime>.ptr])}

<expPrime> ===> TK_EPS
{<expPrime>.ptr = NULL)}

<term> ===> <factor> <termPrime>
{<term>.ptr = newNode([<factor>.ptr, <termPrime>.ptr])}

<termPrime> ===> <highPrecedenceOperators> <factor> <termPrime>
{<termPrime>.ptr = newNode([<highPrecedenceOperators>.ptr, <factor>.ptr,
<termPrime>.ptr])}

<termPrime> ===> TK_EPS
{<termPrime>.ptr = NULL}

<factor> ===> TK_OP <arithmeticExpression> TK_CL
{<factor>.ptr = <arithmeticExpression>.ptr}

<factor> ===> <allVar>
{<factor>.ptr = <allVar>.ptr}

<highPrecedenceOperators> ===> TK_MUL
{<highPrecedenceOperators>.ptr = makeLeaf("TK_MUL", "*")}

<highPrecedenceOperators> ===> TK_DIV
{<highPrecedenceOperators>.ptr = makeLeaf("TK_DIV", "+")}

<lowPrecedenceOperators> ===> TK_PLUS
{<highPrecedenceOperators>.ptr = makeLeaf("TK_PLUS", "+")}

<lowPrecedenceOperators> ===> TK_MINUS
{<highPrecedenceOperators>.ptr = makeLeaf("TK_MINUS", "-")}

<booleanExpression> ===> TK_OP <booleanExpression> TK_CL <logicalOp> TK_OP
<booleanExpression> TK_CL
{<booleanExpression>.ptr = newNode([<logicalOp>.ptr, <booleanExpression>.ptr,
<booleanExpression>.ptr])}

<booleanExpression> ===> <var> <relationalOp> <var>
{<booleanExpression>.ptr = newNode([<relationalOp>.ptr, <var>.ptr, <var>.ptr])}

<booleanExpression> ===> TK_NOT TK_OP <booleanExpression> TK_CL
{<booleanExpression>.ptr = newNode([makeLeaf("TK_NOT"),
<booleanExpression>.ptr])}

<var> ===> TK_ID
{<var>.ptr = makeLeaf("TK_ID", id.entry)}

<var> ===> TK_NUM
{<var>.ptr = makeLeaf("TK_NUM", num.val)}

<var> ===> TK_RNUM
{<var>.ptr = makeLeaf("TK_RNUM", rnum.val)}

<logicalOp> ===> TK_AND
{<logicalOp>.ptr = makeLeaf("TK_AND", &&&)}

<logicalOp> ===> TK_OR
{<logicalOp>.ptr = makeLeaf("TK_OR", "@@@")}

<relationalOp> ===> TK_LT
{<relationalOp>.ptr = makeLeaf("TK_LE", "<=")}

<relationalOp> ===> TK_LE
```

```
{<relationalOp>.ptr = makeLeaf("TK_LT", "<")}

<relationalOp> ===> TK_EQ
{<relationalOp>.ptr = makeLeaf("TK_EQ", "==")}

<relationalOp> ===> TK_GT
{<relationalOp>.ptr = makeLeaf("TK_GT", ">")}

<relationalOp> ===> TK_GE
{<relationalOp>.ptr = makeLeaf("TK_GE", ">=")}

<relationalOp> ===> TK_NE
{<relationalOp>.ptr = makeLeaf("TK_NE", "!=")}

<returnStmt> ===> TK_RETURN <optionalReturn> TK_SEM
{<returnStmt>.ptr = <optionalReturn>.ptr}

<optionalReturn> ===> TK_SQL <idList> TK_SQR
{<optionalReturn>.ptr = <idList>.ptr}

<optionalReturn> ===> TK_EPS
{<optionalReturn>.ptr = NULL}

<idList> ===> TK_ID <more_ids>
{<idList>.ptr = <more_ids>.ptr}

<more_ids> ===> TK_COMMA <idList>
{<more_ids>.ptr = <idList>.ptr}

<more_ids> ===> TK_EPS
{<more_ids>.ptr = NULL}
```