ABSTRACT SYNTAX TREE – GROUP 43

Name: Rahul B
ID: 2019A7PS0134P

Name: Asish Juttu
ID: 2019A7PS0039P

Name: Narasimha Gupta Jangala
ID: 2019A7PS0138P

Name: Praneeth Chaitanya Jonnavithula
ID: 2019A7PS1111P

Name: Damargidda Rohan Raj Goud
ID: 2019A7PS0065P

Functions Used

makeNode() - creates a node of the ast, with label specified by the first parameter and children specified by the following parameters.

makeLeaf() - creates a leaf node of the ast.

insertTo() - takes an ast node as input and returns another ast node which is the concatenation of the following nodes specified as parameters to the children of the first node.

addType() - adds the type information of the first parameter in the symbol table.

<PROGRAM> ===>

```
{
      other_functions.node_inh = makeNode(ast_otherfunctions, null);
}
```

<OTHER_FUNCTIONS> <MAIN_FUNCTION>

```
{
      program.node_syn = makeNode(ast_program, other_functions.node_syn,
main_function.node_syn);
}
```

<MAIN_FUNCTION> ===> TK_MAIN <STMTS> TK_END

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
      main_function.node_syn = makeNode(ast_main, stmts.node_syn);
}
```

<OTHER_FUNCTIONS> ===>

```
{
      function.node_inh = other_functions.node_inh;
}
```

<FUNCTION>

```
{
      other_functions1.node_inh = function.node_syn;
}
```

<OTHER_FUNCTIONS1>

```
{
      other_functions.node_syn = other_functions1.node_syn;
}
```

<OTHER_FUNCTIONS> ===> EPSILON

```
{
      other_functions.node_syn = other_functions.node_inh;
}
```

<FUNCTION> ===> TK_FUNID <INPUT_PAR> <OUTPUT_PAR> TK_SEM <STMTS> TK_END

```
{
      function.node_syn = insertTo(function.node_inh,
makeNode(ast_function, tk_funid.getToken(), input_par.node_syn,
output_par.node_syn, stmts.node_syn));
}
```

<INPUT_PAR> ===> TK_INPUT TK_PARAMETER TK_LIST TK_SQL

```
{
      parameter_list.node_inh = makeNode(ast_parameter_list, null);
}
```

<PARAMETER_LIST> TK_SQR

```
{
      input_par.node_syn = parameter_list.node_syn;
}
```

<OUTPUT_PAR> ===> TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL

```
{
      parameter_list.node_inh = makeNode(ast_parameter_list, null);
```

ABSTRACT SYNTAX TREE – GROUP 43

```
}
```
<PARAMETER_LIST> TK_SQR
```
{
      output_par.node_syn = parameter_list.node_syn;
}
```

<OUTPUT_PAR> ===> EPSILON
```
{
      output_par.node_syn = null;
}
```

<PARAMETER_LIST> ===> <DATATYPE> TK_ID
```
{
      addType(tk_id.getToken(), datatype.node_syn);
      remaining_list.node_inh = insertTo(parameter_list.node_inh,
tk_id.getToken());
}
```
<REMAINING_LIST>
```
{
      parameter_list.node_syn = remaining_list.node_syn;
}
```

<DATATYPE> ===> <PRIMITIVE_DATATYPE>
```
{
      datatype.node_syn = primitive_datatype.node_syn;
}
```

<DATATYPE> ===> <CONSTRUCTED_DATATYPE>
```
{
      datatype.node_syn = constructed_datatype.node_syn;
}
```

<PRIMITIVE_DATATYPE> ===> TK_INT
```
{
      primitive_datatype.node_syn = makeLeaf(tk_int.getToken());
}
```

<PRIMITIVE_DATATYPE> ===> TK_REAL
```
{
      primitive_datatype.node_syn = makeLeaf(tk_real.getToken());
}
```

ABSTRACT SYNTAX TREE – GROUP 43

<CONSTRUCTED_DATATYPE> ===> TK_RECORD TK_RUID

```
{
      contructed_datatype.node_syn = makeLeaf(ast_constructed_datatype,
tk_record.getToken(), tk_ruid.getToken());
}
```

<CONSTRUCTED_DATATYPE> ===> TK_UNION TK_RUID

```
{
      contructed_datatype.node_syn = makeLeaf(ast_constructed_datatype,
tk_union.getToken(), tk_ruid.getToken());
}
```

<CONSTRUCTED_DATATYPE> ===> TK_RUID

```
{
      contructed_datatype.node_syn = makeLeaf(ast_constructed_datatype,
tk_ruid.getToken());
}
```

<REMAINING_LIST> ===> TK_COMMA

```
{
      parameter_list.node_inh = remaining_list.node_inh;
}
```

<PARAMETER_LIST>

```
{
      remaining_list.node_syn = parameter_list.node_syn;
}
```

<REMAINING_LIST> ===> EPSILON

```
{
      remaining_list.node_syn = remaining_list.node_inh;
}
```

<STMTS> ===> <TYPE_DEFINITIONS> <DECLARATIONS> <OTHER_STMTS> <RETURN_STMT>

```
{
      stmts.node_syn = makeNode(ast_stmts, type_definitions.node_syn,
declarations.node_syn, other_stmts.node_syn, return_stmt.node_syn);
}
```

<TYPE_DEFINITIONS> ===>

```
{
      actual_or_redefined.node_inh = type_refinitions.node_inh;
```

ABSTRACT SYNTAX TREE – GROUP 43

```
}
```
<ACTUAL_OR_REDEFINED>
```
{
      type_definitions1.node_inh = actual_or_redefined.node_syn;
}
```
<TYPE_DEFINITIONS1>
```
{
      type_definitions.node_syn = type_definitions1.node_syn;
}
```

<TYPE_DEFINITIONS> ===> EPSILON
```
{
      type_definitions.node_syn = type_definitions.node_inh;
}
```

<TYPE_DEFINITION> ===> TK_RECORD TK_RUID
```
{
      addType(tk_ruid.getToken(), tk_record.getToken());
}
```
<FIELD_DEFINITIONS> TK_ENDRECORD
```
{
      type_definition.node_syn =
insertTo(type_definition.node_inh,makeNode(ast_type_definition,
tk_record.getToken(), tk_ruid.getToken(), field_definitions.node_syn));
}
```

<TYPE_DEFINITION> ===> TK_UNION TK_RUID
```
{
      addType(tk_ruid.getToken(), tk_union.getToken());
}
```
<FIELD_DEFINITIONS> TK_ENDUNION
```
{
      type_definition.node_syn=
insertTo(type_definition.node_inh,makeNode(ast_type_definition,
tk_union.getToken(), tk_ruid.getToken(), field_definitions.node_syn));
}
```

<FIELD_DEFINITIONS> ===> <FIELD_DEFINITION1> <FIELD_DEFINITION2>
```
{
      more_fields.node_inh = makeNode(ast_field_definitions,
field_definitions1.node_syn, field_definitions2.node_syn);
}
```
<MORE_FIELDS>

ABSTRACT SYNTAX TREE – GROUP 43

```
{
      field_definitions.node_syn = more_fields.node_syn;
}
```

<FIELD_DEFINITION> ===> TK_TYPE <FIELD_TYPE> TK_COLON TK_FIELDID TK_SEM

```
{
      field_definition.node_syn =
insertTo(field_definition.node_inh,makeNode(ast_field_definition,
field_type.node_syn,tk_fieldid.getToken()));
      addType(tk_fieldid.getToken(), .field_type.node_syn);
}
```

<MORE_FIELDS> ===>

```
{
      field_definition.node_inh = more_fields.node_inh;
}
```

<FIELD_DEFINITION>

```
{
      more_fields1.node_inh = field_definition.node_syn;
}
```

<MORE_FIELDS1>

```
{
      more_fields.node_syn = more_fields1.node_syn;
}
```

<MORE_FIELDS> ===> EPSILON

```
{
      more_fields.node_syn = more_fields.node_inh;
}
```

<DECLARATIONS> ===>

```
{
      declaration.node_inh = declarations.node_inh;
}
```

<DECLARATION>

```
{
      declarations1.node_inh = declaration.node_syn;
}
```

<DECLARATIONS1>

```
{
      declarations.node_syn = declarations1.node_syn;
}
```

ABSTRACT SYNTAX TREE – GROUP 43

<DECLARATIONS> ===> EPSILON

```
{
      declarations.node_syn = declarations.node_inh;
}
```

<DECLARATION> ===> TK_TYPE <DATATYPE> TK_COLON TK_ID

```
{
      addType(tk_id.getToken(), datatype.node_syn);
}
```

<GLOBAL_OR_NOT> TK_SEM

```
{
      declaration.node_syn = insertTo(declaration.node_inh,
makeNode(ast_declaration, datatype.node_syn, tk_id.getToken(),
global_or_not.node_syn);
}
```

<GLOBAL_OR_NOT> ===> TK_COLON TK_GLOBAL

```
{
      global_or_not.node_syn = makeLeaf(tk_global.getToken());
}
```

<GLOBAL_OR_NOT> ===> EPSILON

```
{
      global_or_not.node_syn = null;
}
```

<OTHER_STMTS> ===>

```
{
      stmt.node_inh = other_stmts.node_inh;
}
```

<STMT>

```
{
      other_stmts1.node_inh = stmt.node_syn;
}
```

<OTHERSTMTS1>

```
{
      other_stmts.node_syn = other_stmts1.node_syn;
}
```

<OTHER_STMTS> ===> EPSILON

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
	other_stmts.node_syn = other_stmts.node_inh;
}
```

<STMT> ===> <ASSIGNMENT_STMT>

```
{
	stmt.node_syn = insertTo(stmt.node_inh, assignment_stmt.node_syn);
}
```

<STMT> ===> <ITERATIVE_STMT>

```
{
	stmt.node_syn = insertTo(stmt.node_inh, iterative_stmt.node_syn);
}
```

<STMT> ===> <CONDITIONAL_STMT>

```
{
	stmt.node_syn = insertTo(stmt.node_inh, conditional_stmt.node_syn);
}
```

<STMT> ===> <IO_STMT>

```
{
	stmt.node_syn = insertTo(stmt.node_inh, io_stmt.node_syn);
}
```

<STMT> ===> <FUN_CALL_STMT>

```
{
	stmt.node_syn = insertTo(stmt.node_inh, fun_call_stm.node_syn);
}
```

<ASSIGNMENT_STMT> ===> <SINGLE_OR_REC_ID> TK_ASSIGNOP <ARITHMETIC_EXPRESSION> TK_SEM

```
{
	assignment_stmt.node_syn =
makeNode(ast_assignment_stmt,single_or_rec_id.node_syn,arithmetic_expressi
on.node_syn);
}
```

<SINGLE_OR_REC_ID> ===> TK_ID <OPTION_SINGLE_CONSTRUCTED>

```
{
	Single_or_rec_id.node_syn = makeNode(ast_single_or_rec_id,
tk_id.getToken(),option_single_constructed.node_syn);
```

```
}
```

<FUN_CALL_STMT> ===> <OUTPUT_PARAMETERS> TK_CALL TK_FUNID TK_WITH TK_PARAMETERS <INPUT_PARAMETERS> TK_SEM

```
{
      fun_call_stmt.node_syn =
makeNode(ast_fun_call_stmt,output_parameters.node_syn,tk_funid.getToken(),
input_parameters.node_syn);
}
```

<OUTPUT_PARAMETERS> ===> TK_SQL

```
{
      id_list.node_inh = makeNode(ast_id_list, null);
}
```

<ID_LIST> TK_SQR TK_ASSIGNOP

```
{
      output_parameters.node_syn = id_list.node_syn;
}
```

<OUTPUT_PARAMETERS> ===> EPSILON

```
{
      output_parameter.node_syn = null;
}
```

<INPUT_PARAMETERS> ===> TK_SQL

```
{
      id_list.node_inh = makeNode(ast_id_list, null);
}
```

<ID_LIST> TK_SQR

```
{
      input_parameters.node_syn = id_list.node_syn;
}
```

<ITERATIVE_STMT> ===> TK_WHILE TK_OP <BOOLEAN_EXPRESSION> TK_CL

```
{
      stmt.node_inh = makeNode(ast_other_stmts, null);
}
```

<STMT>

```
{
      other_stmts.node_inh = stmt.node_syn;
}
```

<OTHER_STMTS> TK_ENDWHILE

ABSTRACT SYNTAX TREE – GROUP 43

```
{
      iterative_stmt.node_syn=makeNode(ast_iterative_stmt,boolean_expressi
on.node_syn, other_stmts.node_syn);
}
```

<CONDITIONAL_STMT> ===> TK_IF TK_OP <BOOLEAN_EXPRESSION> TK_CL TK_THEN

```
{
      stmt.node_inh = makeNode(ast_other_stmts, null);
}
```

<STMT>

```
{
      other_stmts.node_inh = stmt.node_syn;
}
```

<OTHER_STMTS> <ELSE_PART>

```
{
      conditional_stmt.node_syn=makeNode(ast_conditional_stmt,boolean_expr
ession.node_syn,other_stmts.node_syn, else_part.node_syn);
}
```

<IO_STMT> ===> TK_READ TK_OP <VAR> TK_CL TK_SEM

```
{
      Io_stmt.node_syn = makeNode(tk_read.getToken(), var.node_syn);
}
```

<IO_STMT> ===> TK_WRITE TK_OP <VAR> TK_CL TK_SEM

```
{
      Io_stmt.node_syn = makeNode(tk_write.getToken(), var.node_syn);
}
```

<ARITHMETIC_EXPRESSION> ===> <TERM>

```
{
      exp_prime.node_inh = term.node_syn;
}
```

<EXP_PRIME>

```
{
      arithmetic_expression.node_syn = exp_prime.node_syn;
}
```

<BOOLEAN_EXPRESSION> ===> TK_OP <BOOLEAN_EXPRESSION1> TK_CL
<LOGICAL_OP> TK_OP <BOOLEAN_EXPRESSION2> TK_CL

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
	boolean_expression.node_syn =
makeNode(logical_op.node_syn.getLabel(), boolean_expression1.node_syn,
boolean_expression2.node_syn);
}
```

<BOOLEAN_EXPRESSION> ===> <VAR> <RELATIONAL_OP> <VAR1>

```
{
	boolean_expression.node_syn =
makeNode(relational_op.node_syn.getLabel(), var.node_syn, var1.node_syn;
}
```

<BOOLEAN_EXPRESSION> ===> TK_NOT TK_OP <BOOLEAN_EXPRESSION1> TK_CL

```
{
	boolean_expression.node_syn = makeNode(ast_not,
boolean_expression1.node_syn);
}
```

<VAR> ===> <SINGLE_OR_REC_ID>

```
{
	var.node_syn = single_or_rec_id.node_syn;
}
```

<VAR> ===> TK_NUM

```
{
	var.node_syn = makeLeaf(ast_num, tk_num.getToken());
}
```

<VAR> ===> TK_RNUM

```
{
	var.node_syn = makeLeaf(ast_rnum, tk_rnum.getToken());
}
```

<LOGICAL_OP> ===> TK_AND

```
{
	logical_op.node_syn = makeLeaf(ast_and);
}
```

<LOGICAL_OP> ===> TK_OR

```
{
	logical_op.node_syn = makeLeaf(ast_or);
```

```
}
```

<RELATIONAL_OP> ===> TK_LT

```
{
      relational_op.node_syn = makeLeaf(ast_lt);
}
```

<RELATIONAL_OP> ===> TK_LE

```
{
      relational_op.node_syn = makeLeaf(ast_le);
}
```

<RELATIONAL_OP> ===> TK_EQ

```
{
      relational_op.node_syn = makeLeaf(ast_eq);
}
```

<RELATIONAL_OP> ===> TK_GT

```
{
      relational_op.node_syn = makeLeaf(ast_gt);
}
```

<RELATIONAL_OP> ===> TK_GE

```
{
      relational_op.node_syn = makeLeaf(ast_ge);
}
```

<RELATIONAL_OP> ===> TK_NE

```
{
      relational_op.node_syn = makeLeaf(ast_ne);
}
```

<RETURN_STMT> ===> TK_RETURN <OPTIONAL_RETURN> TK_SEM

```
{
      return_stmt.node_syn = optional_return.node_syn;
}
```

<OPTIONAL_RETURN> ===> TK_SQL

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
      id_list.node_inh = makeNode(ast_id_list, null);
}
```
<ID_LIST> TK_SQR
```
{
      optional_return.node_syn = id_list.node_syn;
}
```

<OPTIONAL_RETURN> ===> EPSILON
```
{
      optional_return.node_syn = null;
}
```

<ID_LIST> ===> TK_ID
```
{
      more_ids.node_inh = insertTo(id_list.node_inh, tk_id.getToken());
}
```
<MORE_IDS>
```
{
      id_list.node_syn = more_ids.node_syn;
}
```

<MORE_IDS> ===> TK_COMMA
```
{
      id_list.node_inh = more_ids.node_inh;
}
```
<ID_LIST>
```
{
      more_ids.node_syn = id_list.node_syn;
}
```

<MORE_IDS> ===> EPSILON
```
{
      more_ids.node_syn = more_ids.node_inh;
}
```

<ACTUAL_OR_REDEFINED> ===> <TYPE_DEFINITION>
```
{
      actual_or_redefined.node_syn =
insertTo(actual_or_redefined.node_inh, type_definition.node_syn);
}
```

ABSTRACT SYNTAX TREE – GROUP 43

<**ACTUAL_OR_REDEFINED> ===>** <DEFINE_TYPE_STATEMENT>

```
{
      actual_or_redefined.node_syn =
insertTo(actual_or_redefined.node_inh, definite_type_statement.node_syn);
}
```

<DEFINITE_TYPE_STATEMENT> ===> TK_DEFINETYPE <A> TK_RUID TK_AS TK_RUID1

```
 {
      definite_type_statement.node_syn = makeNode(ast_define_type_stmt,
a.node_syn, tk_ruid.getToken(), tk_ruid1.getToken());
}
```

<FIELD_TYPE>===> <PRIMITIVE_DATATYPE>

```
{
      field_type.node_syn = primitive_datatype.node_syn;
}
```

<FIELD_TYPE>===>TK_RUID

```
{
      field_type.node_syn = makeLeaf(tk_ruid.getToken());
}
```

<OPTION_SINGLE_CONSTRUCTED>===>

```
{
      one_expansion.node_inh = makeNode(ast_more_expansion, null);
}
```

<ONE_EXPANSION> <MORE_EXPANSIONS>

```
{
      option_single_constructed.node_syn =
makeNode(ast_option_single_constructed, more_expansions.node_syn);
}
```

<OPTION_SINGLE_CONSTRUCTED>===>EPSILON

```
{
      option_single_constructed.node_syn = null;
}
```

<ONE_EXPANSION>===>TK-DOT TK_FIELDID

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
      one_expansion.node_syn = insertTo(one_expansion.node_inh,
tk_field_id.getToken());
}
```

<MORE_EXPANSION>===>

```
{
      one_expansion.node_inh = more_expansions.node_inh;
}
```

<ONE_EXPANSION>

```
{
      more_expansions1.node_inh = one_expansion.node_syn;
}
```

<MORE_EXPANSION>

```
{
      more_expansions.node_syn = more_expansions1.node_syn;
}
```

<MORE_EXPANSION> ===> EPSILON

```
{
      more_expansions.node_syn = more_expansions.node_inh;
}
```

<A> ===> TK_RECORD

```
{
      a.node_syn = makeLeaf(ast_a, tk_record.getToken());
}
```

<A> ===> TK_UNION

```
{
      a.node_syn = makeLeaf(ast_a, tk_union.getToken());
}
```

<ELSE_PART>===>TK_ELSE

```
{
      stmt.node_inh = makeNode(ast_other_stmts, null);
}
```

<STMT>

```
{
      other_stmts.node_inh = stmt.node_syn;
}
```

<OTHER_STMTS> TK_ENDIF

ABSTRACT SYNTAX TREE – GROUP 43

```
{
      else_part.node_syn=makeNode(ast_else_part,other_stmts.node_syn);
}
```

<ELSE_PART>===>TK_ENDIF

```
{
      else_part.node_syn = makeLeaf(tk_endif.getToken());
}
```

<TERM>===>

```
{
      factor.node_inh = makeNode(ast_factor, null);
}
```

<FACTOR> <TERM_PRIME>

```
{
      term.node_syn = makeNode(ast_term, factor.node_syn,
term_prime.node_syn);
}
```

<EXP_PRIME>===><LOW_PRECEDENCE_OPERATORS> <TERM>

```
{
      exp_prime1.node_inh =
makeNode(low_precedence_operator.node_syn.getLabel(), exp_prime.node_inh,
term.node_syn);
}
```

<EXP_PRIME1>

```
{
      exp_prime.node_syn = exp_prime1.node_syn;
}
```

<EXP_PRIME>===> EPSILON

```
{
      exp_prime.node_syn = exp_prime.node_inh;
}
```

<LOW_PRECEDENCE_OPERATORS> ===> TK_PLUS

```
{
      low_precedence_operators.node_syn = makeLeaf(ast_plus);
}
```

<LOW_PRECEDENCE_OPERATORS> ===> TK_MINUS

```
{
```

ABSTRACT SYNTAX TREE – GROUP 43

```
        low_precedence_operators.node_syn = makeLeaf(ast_minus);
}
```

<FACTOR> ===> TK_OP <ARITHMETIC_EXPRESSION>

```
{
        factor.node_syn = arithmetic_expression.node_syn;
}
```

TK_CL

<FACTOR> ===> <VAR>

```
{
        factor.node_syn = var.node_syn;
}
```

<TERM_PRIME> ===> <HIGH_PRECEDENCE_OPERATORS> <FACTOR>

```
{
        term_prime1.node_inh =
makenode(high_precedence_operator.node_syn.getLabel(),
term_prime.node_inh, factor.node_syn);
}
```

<TERM_PRIME>

```
{
        term_prime.node_syn = tech_prime1.node_syn;
}
```

<TERM_PRIME> ===> EPSILON

```
{
term_prime.node_syn = term_prime.node_inh;
}
```

<HIGH_PRECEDENCE_OPERATORS> ===> TK_MUL

```
{
        high_precedence_operators.node_syn = makeleaf(ast_mul);
}
```

<HIGH_PRECEDENCE_OPERATORS> ===> TK_DIV

```
{
        high_precedence_operators.node_syn = makeleaf(ast_div);
}
```

ABSTRACT SYNTAX TREE – GROUP 43