Assignment 04

Compiler

Group _04

# Team Members: Tejendra Khatri, Sushil Pandey, Ankur Lamichhane

Purpose of the assignment: To extend our grammar from previous assignment and build the abstract syntax tree for extended grammar.

All the requirements given in the assignment is completely done. Class Lexer reads the input from the text file and class parser extends ASTVisitor. Parser then generate the abstract syntax tree based on the input that Lexer read from textfile. Necessary node classes are added in parser file directory. Unparser is also implemented. Unparser will print the program on out file (in output.txt) in with proper convention.

In the assignment, as unary increment and unary decrement were optional, we did not implement in this assignment. So, in for loop our grammar does not accept the unary increment and unary decrement. That is why, it should be assigned using operator like i= i + 1 or i = i -1. Otherwise, all operators mentioned in assignment are implemented and the program also follows the order of the precedence.

Parenthesis (expr) is also implemented in the program. If statement and for statement with optional initialization, optional logical expression and update expression are also implemented in the assignment. As we mention earlier, for update expression in for statement, we should assign like i=i+1 (++ and -- operators were optional, so we have not implemented them in this stage). They shall be implemented next time along with the unary operator.

Roles and Responsibilities:

As in earlier assignment, we studied separately and later discussed about possible ways to accomplish the assignment. Basically, Tejendra played vital in coding role and Sushil, and Ankur, helped inside coding and debugging along with grammar and pdf file. So, all have played significant role in the completion of the project.

Group_04_Compiler_Assign04_Grammar:

CompilationUnit Node ➔BlockNode

BlockNode➔ ‘{’DeclarationNode StatementNode ‘}’

DeclarationNode➔TypeNode LiteralNode | e

StatementNode➔AssignmentNode | IfNode | ForNode| e

AssignmentNode➔LiteralNode ‘=’ ExpressionNode

ExpressionNode ➔valueNode | BinaryExpressionNode
                    | (ExpressionNode)

ValueNode➔LiteralNode

BinaryExpressionNode➔LiteralNode Operator LiteralNode
                    | LiteralNode Operator ExpressionNode
                    | ExpressionNode Operator LiteralNode

Operator ➔ * | / | + | -

IfNode ➔if (LogicalExpressionNode) BlockNode [else BlockNode]

LogicalExpressionNode ➔ LiteralNode Operation LiteralNode

Operation ➔ = = | != | > | >= | < | <=

ForNode ➔ for(; ;)BlockNode
                    | for(AssignmentNode; LogicalExpressionNode; ForUpdateExpressionNode) BlockNode

ForUpdateExpressionNode➔LiteralNode ‘=’ BinaryExpressionNode