Assignment 06 Compilers

Team members:

- Tejendra Khatri
- Sushil Pandey
- Ankur Lamichhane

Roles and Responsibilities:

Although it was a group effort, due to differences in ideas and working strategy, everybody did their coding on their own helping each other to find and fix bugs. However, the language specification has been formed by the combined effort of everyone. Finally, we jotted down the working parts from everybody's code and then came up with our final product.

Changes made to the language in this program:

We started implementing the type Char but later realized it was not required, so it is left incomplete.

Parts completed for Char: can only be declared. No initialization or use.

Symbol table has been implemented using the class Env. Since the whole program is inside a block we decided the following rules:

- The same variable may not be declared twice.
- Once a variable has been declared, it can be used anywhere after the declaration statement;

Added the code to directly initialize a Boolean variable to true or false.

Previous code:	New code:
Boolean x;	Boolean x;
X = true; //not allowed	x = true; //allowed
X = 2==2; //allowed	x = 2==2; //allowed

Unary minus, unary increment and unary decrement have not been implemented for simplicity since these statements can always be replaced by assignment statements. So we decided not to implement in our grammar. So, I removed the code in the lexer where the lexer could read the ++ and – operators.

TypeChecker:

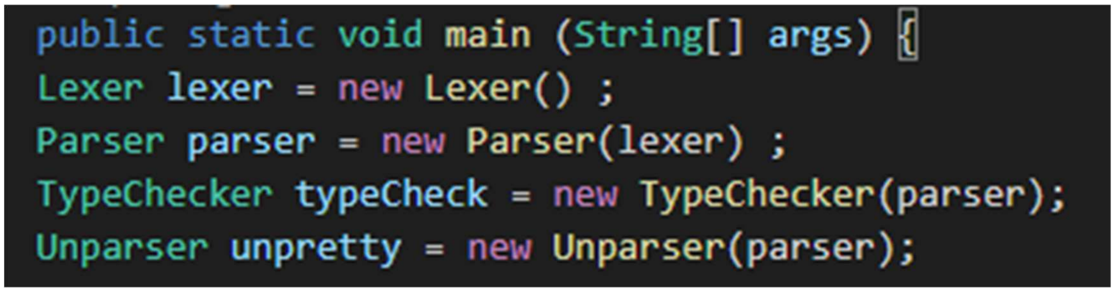
A class TypeChecker has been created to implement type checking in the program. It is a subclass of ASTVisitor. It helps to check the errors that the parser was unable to detect. It throws errors corresponding to respective error type. Note that narrow as well as widening conversions have been implemented.

Basic tasks done by this class are as follows:

1. For a declaration node, it checks to see if the variable being declared has been declared before.
2. For an assignment node, it checks to see that the variable being assigned and the value being assigned is of the same type or equivalent.
3. For an expression node where the operations are only addition, subtraction, multiplication and division, it checks to see no Booleans are present in the expression.
4. For an If node, it checks to that the type of logical expression node is a Boolean.

These are some of the tasks done by our TypeChecker.

Here is a snapshot of main:



```
public static void main (String[] args) {
Lexer lexer = new Lexer() ;
Parser parser = new Parser(lexer) ;
TypeChecker typeCheck = new TypeChecker(parser);
Unparser unpretty = new Unparser(parser);
```

Here, we call parser first to build the AST. After that, we traverse through the tree performing type checking. If and only if all the types match then the Unparser unpretty produces the output.txt file else output.txt is not created.