# **Project: Boolean Expression Evaluator**

Vijaya R Bankar

vbankar@my.harrisburgu.edu

Harrisburg University of Science and Technology

Course...section

**Abstract**:

Compiler is a program that converts high level program into lower-level form so that they can be read and executed by a program. The compiler which I designed will read Ternary (or Boolean) expression, parse and execute it. It is a Boolean Expression Evaluator. The expressions (or statements) are divided into tokens. Parse tree is created from tokens and each node of the tree is visited. The expressions are evaluated as and when each node is visited.

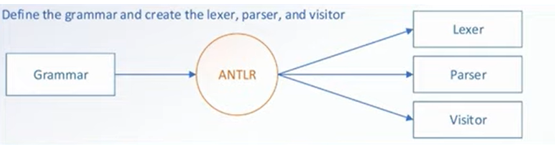
**Design**:

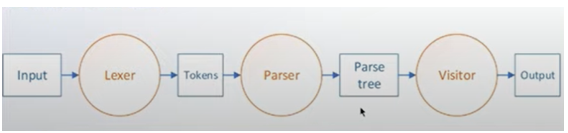
The important part of the design is to create the rule. Every language (be it English or Java) has rules or grammar.

The Compiler is divided into three components: Lexer, Parser and Visitor. The first step is to prepare the grammar file (.g4 file). Like every language, we have grammar file too. It has set of Lexer and Parser rules. The lexer rules define the syntax and parser defines semantic rules. ANTLR tool (creates lexer and parser from .g4 file) takes .g4 file as input and generated three components which I mentioned earlier. Lexer, Parser and Visitor. These are one time generated. These are Java files generated by ANTLR tool.

For each input file, the ANTLR tool runs the lexer module (java program) first. The lexer creates tokes. This token stream is passed to Parser module. This module creates parser tree. The final module is the Visitor. This is an interface, and we implement the needed methods. The ANTRL tool visits the nodes of the parse tree.

**Pictorial representation**:





**Grammar**:

Grammar is set of rules. Each statement should be well defined. For example, a statement could be an assignment or if statement or log statement.

stat

: assignment

| if\_stat

| log

;

Each of these should be defined. If stat is a node, assignment, if\_state and log are its child nodes.

Stat

Assignment if\_stat log

grammar condition;

parse

: block EOF

;

block

: stat\*

;

stat

: assignment

| if\_stat

| log

;

assignment

: ID ASSIGN expr SCOL

| ID ASSIGN if\_stat SCOL

;

if\_stat

: condition '?' true\_block ':' false\_block

;

condition

: expr

;

true\_block

: atom

;

false\_block

: atom

;

log

: LOG expr SCOL

;

expr

: expr op=(LTEQ | GTEQ | LT | GT) expr #relationalExpr

| expr op=(EQ | NEQ) expr #equalityExpr

| expr AND expr #andExpr

| expr OR expr #orExpr

| atom #atomExpr

;

atom

: OPAR expr CPAR #parExpr

| (INT) #numberAtom

| (TRUE | FALSE) #booleanAtom

| ID #idAtom

;

OR : '||';

AND : '&&';

EQ : '==';

NEQ : '!=';

GT : '>';

LT : '<';

GTEQ : '>=';

LTEQ : '<=';

SCOL : ';';

ASSIGN : '=';

OPAR : '(';

CPAR : ')';

OBRACE : '{';

CBRACE : '}';

TRUE : 'true';

FALSE : 'false';

NIL : 'nil';

IF : 'if';

ELSE : 'else';

LOG : 'log';

ID

: [a-zA-Z\_] [a-zA-Z\_0-9]\*

;

INT

: [0-9]+

;

SPACE

: [ \t\r\n] -> skip

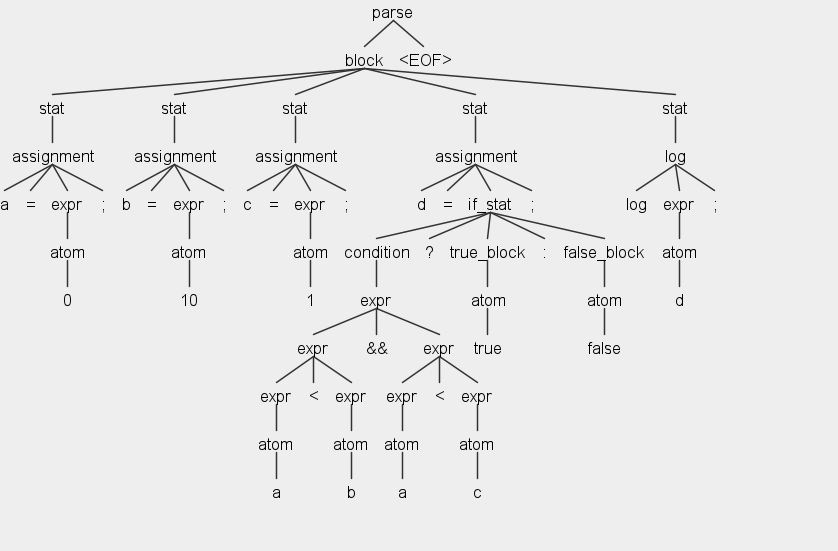
;

For the code below, the following tree would get generated.

a = 0; b = 10; c = 1;

d = a < b && a < c ? true : false;

log d;



**Implementation details:**

Programming language: Java

Tools: ANTLR

Build tool: Maven

IDE: Eclipse

Repository: GIT

Project structure: The compiler is a maven project. Pom.xml has dependencies needed for running the compiler. Artifact antlr4 downloads needed dependencies from maven repository and antlr4-maven-plugin generates java files from .g4 file. When we do maven update, the plug in kicks in and generates needed files.

<dependencies>

<dependency>

<groupId>org.antlr</groupId>

<artifactId>antlr4</artifactId>

<version>4.7.1</version>

</dependency>

</dependencies>

<plugin>

<groupId>org.antlr</groupId>

<artifactId>antlr4-maven-plugin</artifactId>

<version>4.7.1</version>

<executions>

<execution>

<goals>

<goal>antlr4</goal>

</goals>

<configuration>

<visitor>true</visitor>

</configuration>

</execution>

</executions>

</plugin>

Once the files are generated, these files are placed in proper package structure and are compiled. The compiled files are present in jar file.

ANTLR generated files: ANTLR generated files have names with name of the grammar files as prefix. For example, if the name of the grammar file is condition.g4, ANTLR generates the below files:

conditionBaseVisitor.java

conditionLexer.java

conditionParser.java and so on…

Main java program:

When we run the main (HelloWorld.java), it reads the input file demo.hu from resource folder. The file is read line by line and the concatenated string is passed as input to Lexer

conditionLexer lexer = **new** conditionLexer(CharStreams.*fromString*(content.toString()));

The lexer is then passed to parser:

conditionParser parser = **new** conditionParser(**new** CommonTokenStream(lexer));

The next step is to create a tree:

ParseTree tree = parser.parse();

Now, its time to visit the nodes of the parse tree:

conditionBaseVisitor visitor = new conditionBaseVisitor();

visitor.visit(tree);

We don’t make any changes to any of the generated files. We override some of the needed methods in conditionBaseVisitor.java. For example, we override

@Override **public** Value visitAssignment(conditionParser.AssignmentContext ctx) {

**if** (ctx.if\_stat() != **null**) {

String id = ctx.ID().getText();

Value value = **this**.visit(ctx.if\_stat());

**return** memory.put(id, value);

} **else** {

String id = ctx.ID().getText();

Value value = **this**.visit(ctx.expr());

**return** memory.put(id, value);

}

}

ANTLR invokes the method visitAssignment method when it sees any assignment like a = 1;

It gives the context of that node as input. We will have access to the node and its children and we visit the other nodes accordingly. If the assignment is like d = a >b ? 1: 0, ctx.if\_stat() is not null and the if block is executed. For statement a =1, the else block is executed. The left side of the statement is key in hash map and value is result of evaluation.

**Github link**: <https://github.com/bankarvij/compiler_vijaya>

**environment setup & compilation instruction:** Please refer to Project setup document.

**Youtube links**:

<https://youtu.be/-g6R2O0X-L8> (Part 1)

<https://youtu.be/BX_BoHPWrrU> (Part 2)

**References**: <https://www.antlr.org/>