**ASSIGNMENT**

**Mathematical Operations and Conditional Statements using Grammar**

Chinmay Pradeep Kelkar – ckelkar@my.harrisburgu.edu

Harrisburg University of Science and Technology

**Abstract**

Lexers and Parsers are great tools to convert any input to any target programming language. With The help of lexer rules and parsers rules, any input which fits into those rules, can get converted to any language. The input of Lexer gets converted into tokens and these tokens function as an input to the Parser which creates parse tree. The parse trees are used to evaluate the input.

Writing your own lexers and parsers is little time consuming, and you have to maintain your code as language you write yours lexers and parsers in, continuously evolves. Taking all these into consideration, ANTLR is a great tool to generate lexers and parsers. Given a grammar in .g4 file, ANTLR generates Lexer and Parser for you. The output of Parser is Abstract Syntax tree. ANTLR also generates visitor and listener which are used to walk through the generated AST and eventually get desired output.

Project Goal:

The project aims at creating simple calculator which handles operations like addition, subtraction, multiplication, division. It uses parentheses to give certain operator higher propriety. Calculator also handles trigonometric operations like sin, cos, tan, cot. It is also capable of handling square root, square, log to the base 10.

Design:

A picture containing text, full

Description automatically generated

Grammer file is provided to ANTLR which generates Lexer, Parser and Visitor. The driver calls functions in above generated files to create tokens, create parser and walk over tree to generate output. User input is provided through .txt file.

Implementation Details:

* For this project, I have created a Net 5.0 console application. I used JetBrains rider to handle all coding. I installed ANTLR v4 plugin in rider which helped me in syntax highlighting, error checking, creating lexers and parsers. I installed Antlr4.Runtime.Standard from nuget to the console project.
* Calculator.g4 contains all the grammar and it is divided into two parts. Lexer rules contains all the rules which are required to generate tokens from input. Parser rules contains all the rules which are required to create syntax tree.
* After configuration of ANTLR, the above grammar generates lexer, parser and visitor.
* User input is provided using input.txt file which contains different mathematical expressions. Driver program loops through all mathematical expressions. For each expression, tokens are generated. Those tokens are provided to parser which generates abstract syntax tree and visitor walks through AST to evaluate the expression.
* Each parser rule generates its respective functions which Calculate.cs uses to implement required calculator. Calculate.cs extends calculateBaseVisitor class and implements all the functions. VisitStart is developed from the start rule and can be treated as starter function which starts evaluating expression. Each function has respective context as an input. E.g VisitAddSubtract() has addsubtractcontext as an input.
* Calculate.cs extends calculateBaseVisitor class which implements all the functions required for calculator. VisitStart can be treated as starter function which starts evaluating expression. Each function has respective context as an input. E.g VisitAddSubtract() has addsubtractcontext as an input. Each function calls visit function until it hits leaf node i.e Number.
* The operators with same priority are handled in same function. E.g VisitAddSubtract() handles add and subtract operations as they have same priority.

Grammar:

The grammar consists of two sections.

* Lexer rules: These are used to recognize tokens. Tokens can be categorized as keywords and operators.

A picture containing background pattern

Description automatically generated

e.g if input is “sqrt 16”, “sqrt” can be recognized using SQUAREROOT rule. NUMBER gets evaluated as either INTEGER or FLOAT number. INTEGER and FLOAT can be recognized using their respective regular expressions.

* Parser rules: These defines rules which are used to parse expression. It contains different rule to parse add, subtract, divide, multiply, log, trigonometric functions etc. ANTLR handles precedence of operators in top-down manner. Therefore, parens rule comes at the top and multiplication/divison comes second.

Text

Description automatically generated

e.g the expression 4\*(5+3) gets evaluated as follows.

Graphical user interface, application, Teams

Description automatically generated

According to operator precedence, parens gets evaluated first then multiplication gets applied to the result of parens.

Environment setup:

1. Download dotnet 5 sdk which is suitable to the operating system from [here](https://dotnet.microsoft.com/download/dotnet/5.0)
2. Clone code locally from [here](https://github.com/chinmaykelkar9141/TOC-final-project)

Compilation instructions:

1. After cloning the code, navigate to /FinalProject folder
2. To build : “dotnet build”
3. To run : “dotnet run”

**Reference**

1. Durham, C. (2019, October 12). What is a Lexer, Anyway? DEV Community. <https://dev.to/cad97/what-is-a-lexer-anyway-4kdo>

2. A. (n.d.). antlr/intellij-plugin-v4. GitHub. https://github.com/antlr/intellij-plugin-v4

3. Tomassetti, G. (2021, March 20). The ANTLR Mega Tutorial. Strumenta. https://tomassetti.me/antlr-mega-tutorial/#chapter23

4*.* Create a Text Parser in C# with ANTLR. (2020, November 12). Channel 9. https://channel9.msdn.com/Events/dotnetConf/2020/Create-a-Text-Parser-in-C-with-ANTLR