
Table of Contents

Introduction.....	1
Objective	2
Setting up to build Calculator.	2
Requirements	2
Installation process.....	2
TOOLS AND METHODS.....	3
My Calculator	3
Symbol declarations.....	3
Grammar Rules	3
Grammar in NetBeans.....	4
Lexical Analyzer	6
Output Screen.....	7
Output after running the program (my calculator).....	9
Calculator Screen.....	9
Infix Input	10
Input with parentheses ‘ (‘ and ‘) ‘	10
Prefix Input	11
Postfix Input.....	11
Syntax error.....	12
Conclusion	12
Reference	13

Introduction

JFlex is a lexical analyzer generator (also known as scanner generator) for Java, written in Java.

A lexical analyzer generator takes as input a specification with a set of regular expressions and corresponding actions. It generates a program (a lexer) that reads input, matches the input against the regular expressions in the spec file, and runs the corresponding action if a regular expression matched. Lexers usually are the first front-end step in compilers, matching keywords, comments, operators, etc., and generating an input token stream for parsers. Lexers can also be used for many other purposes.

JFlex supports JDK 1.8 or above for build and JDK 7 and above for run-time.

JFlex Lexers are based on deterministic finite automata (DFAs). They are fast, without expensive backtracking. A standard tool for development of lexical analyzers is **JFlex**.

JFlex is designed to work together with the LALR parser generator CUP, and the Java modification. It can also be used together with other parser generators like ANTLR (ANother Tool for Language Recognition) or as a standalone tool.

A JFlex program consists of three parts:

1. *User code*
 2. *Options, Declarations and Definitions*
 3. *Translation rules*
- ✓ **User code:** is copied verbatim into the beginning of the Java source file of the generated lexer. This is the place for *package* declaration and *import* statements.
 - ✓ **Options:** customize the generated lexer, declare constants, variables, regular definitions
 - ✓ **Translation rules:** have the form $p \{ action \}$ where p is a regular expression and action is Java code specifying what the lexer executes when a sequence of characters of the input string matches p .

The lexical analyzer created by JFlex works as follows:

When activated, the lexical analyzer reads the remaining input one character at a time, until it has found the longest prefix that is matched by one of the regular expressions on the left-hand side of the translation rules.

Objective

This work was done as a partial fulfilment of course Programming Languages and Compilers (AT70.07). The main objective of this assignment is to create a simple calculator that supports addition and multiplication operation, and to know the key compiler concepts like lexical analyzers, semantic analysis, and parse trees.

Setting up to build Calculator.

The Ubuntu desktop is easy to use, easy to install and includes everything we need to run our work. It is also open source, secure, accessible, and free to download.

Requirements

We need to consider the following before starting the installation:

- Ensure you have at least 25 GB of free storage space, or 5 GB for a minimal installation.
- Have access to either a DVD or a USB flash drive containing the version of Ubuntu you want to install.
- Make sure we have a recent backup of our data. While it's unlikely that anything will go wrong.

Installation process

1. Once we have minimum hardware and software requirements follow the link given below:

<https://ubuntu.com/tutorials/install-ubuntu-desktop#5-prepare-to-install-ubuntu>

2. If we want to install Ubuntu on Virtual Box, then follow the link given below:

<https://brb.nci.nih.gov/seqtools/installUbuntu.html>

3. Once we have installed Ubuntu in our system then we need to install NetBean as it is open-source Integrated Development Environment (IDE).

For this assignment I have install the Ubuntu on Virtual Box since I can manage two Operating System and installation was based on Ubuntu 20.04.

Two more steps to follow to install NetBean:

1. Install openjdk 1.8.0 (<https://openjdk.java.net/install/> - This link will help us to understand the process)

sudo apt-get install openjdk-8-jdk

2. Install Netbean 8.0.2 (<https://download.netbeans.org/netbeans/8.0.2/final/> - This link helps us to use the NetBean and how it is configured)

I used Java EE and run the installation script

Since now there are new version of NetBeans and its not compatible with our cup file when I tried. Therefore, I would suggest using the old version of NetBeans which we can get from following link:

<https://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-3413139-esa.html>

TOOLS AND METHODS

The following table briefly shows the software and tools used for developing this calculator.

Name	Description
JFlex	A lexical analyzer generator tool based on Java
CUP	A LALR parser generator that works together with JFlex
NetBeans	IDE for Java development
Java Swing	Framework used to build GUI for the application

My Calculator

The assignment was to develop a calculator with Addition and Multiplication operator. *The Calculator should be able to take integer input and compute the operation and display it. If the input is infix, then output should be value of operation and show the prefix and postfix expression of it.* Even if the input is prefix or postfix then the calculator should be able show the infix expression and vice versa.

Symbol declarations

Terminal: PLUS, MULT, OPAREN – (, CPAREN –) , NUMBER

Nonterminal of Parse Tree are: S, E, T, F, K, L, J, M

Grammar Rules

Rule implemented in CUP file to have Infix, Prefix and Postfix operation of Plus (+) and Multiplication (*) is as given below:

Infix Rule:

$$S = E$$

$$E = E + T \mid T$$

$$T = T * F \mid F$$

$$F = (E) \mid \text{NUMBER}$$
Postfix Rule:

$$S = L$$

$$L = K K + \mid K K *$$

$$K = L$$

$$K = \text{NUMBER}$$
Prefix Rule:

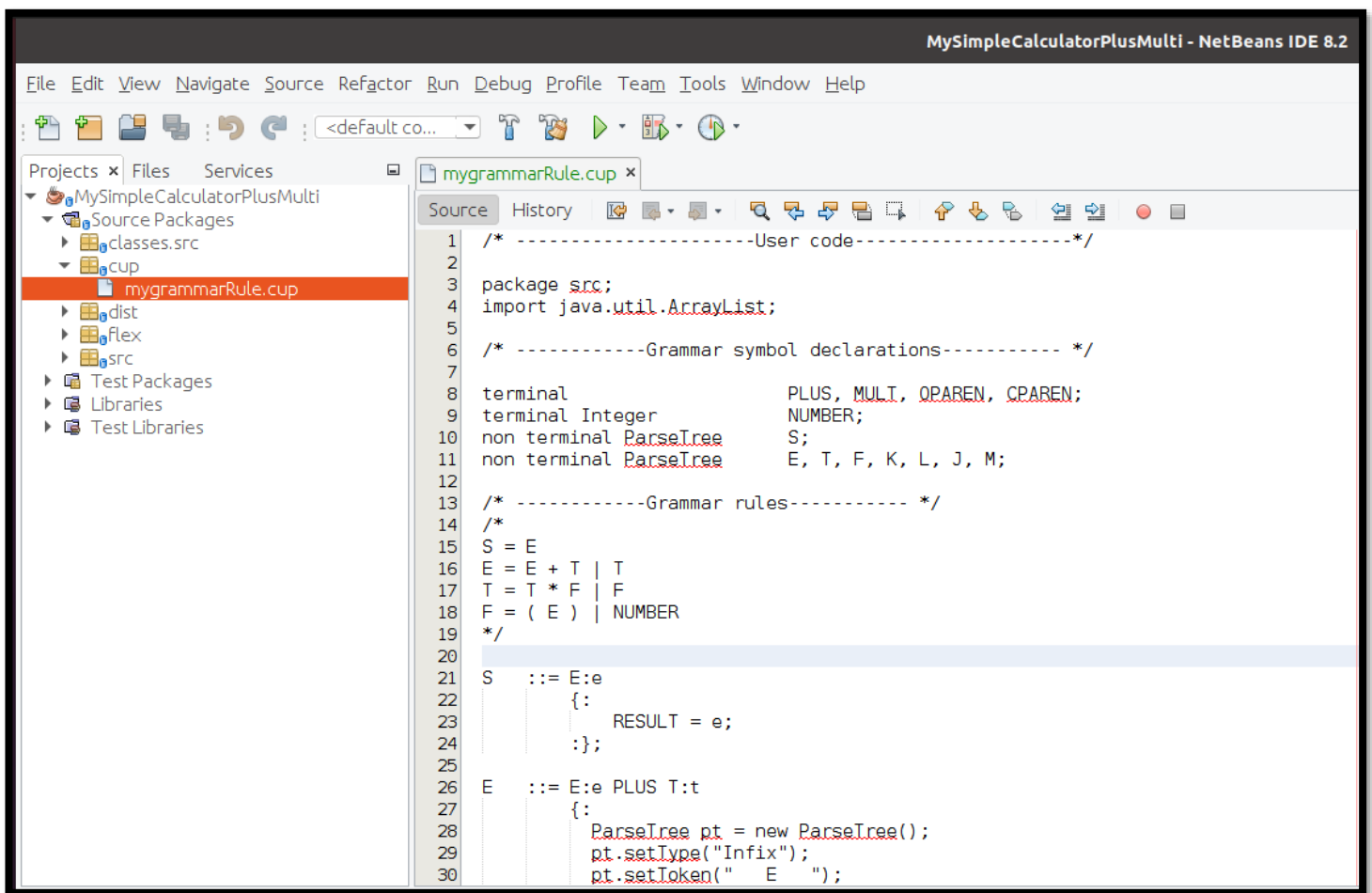
$$S = M$$

$$M = + J J \mid * J J$$

$$J = M$$

$$J = \text{NUMBER}$$

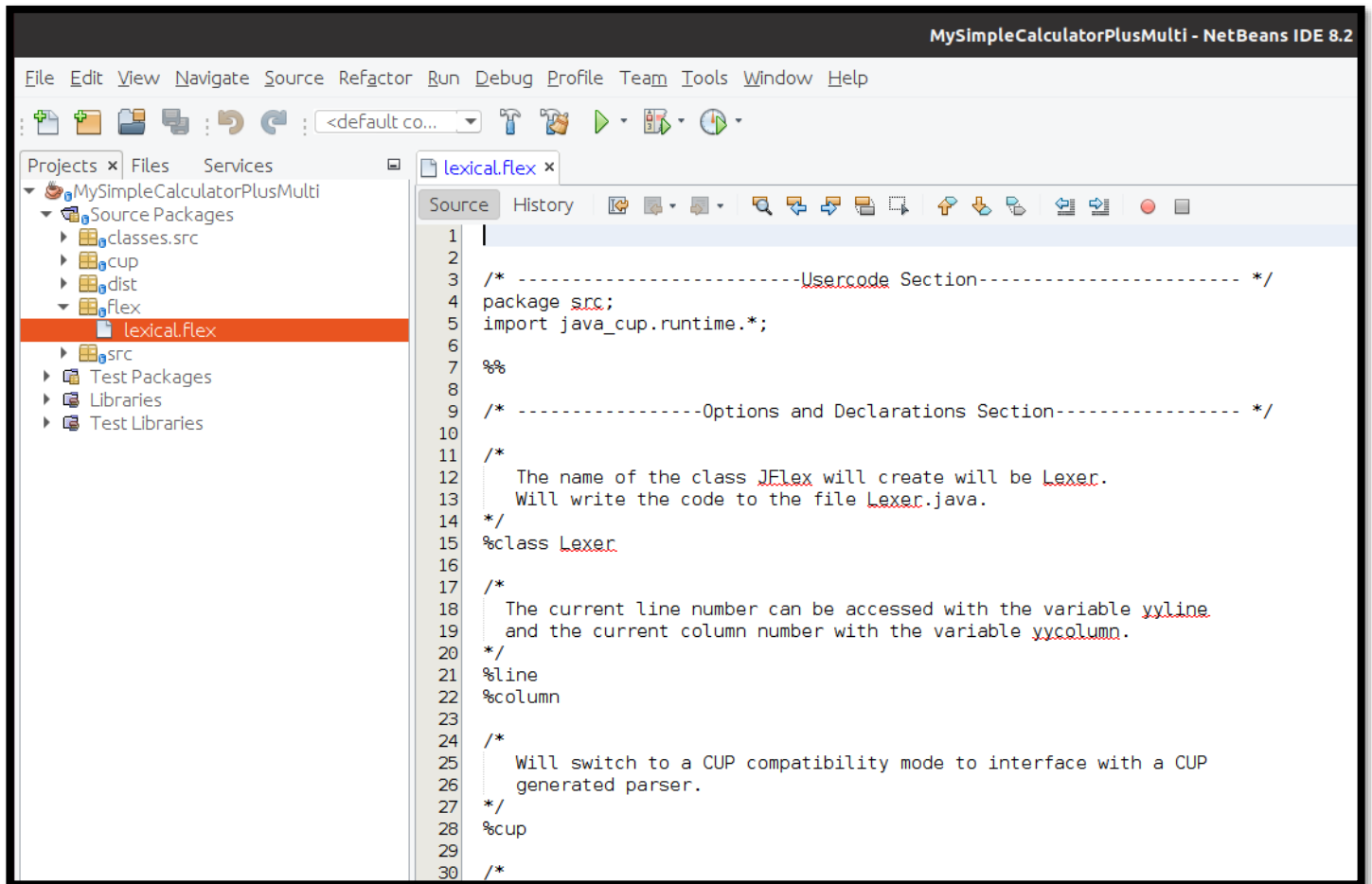
Grammar in NetBeans



The above figure shows that I have **MySimpleCalculatorPlusMulti** project where I have my grammar rule in cup file.

In the **mygrammarRule.cup**, you can see the rule for parse tree like what we have learned from class. The grammar consist of Infix, Postfix and Prefix notation for addition and multiplication operation.

Lexical Analyzer



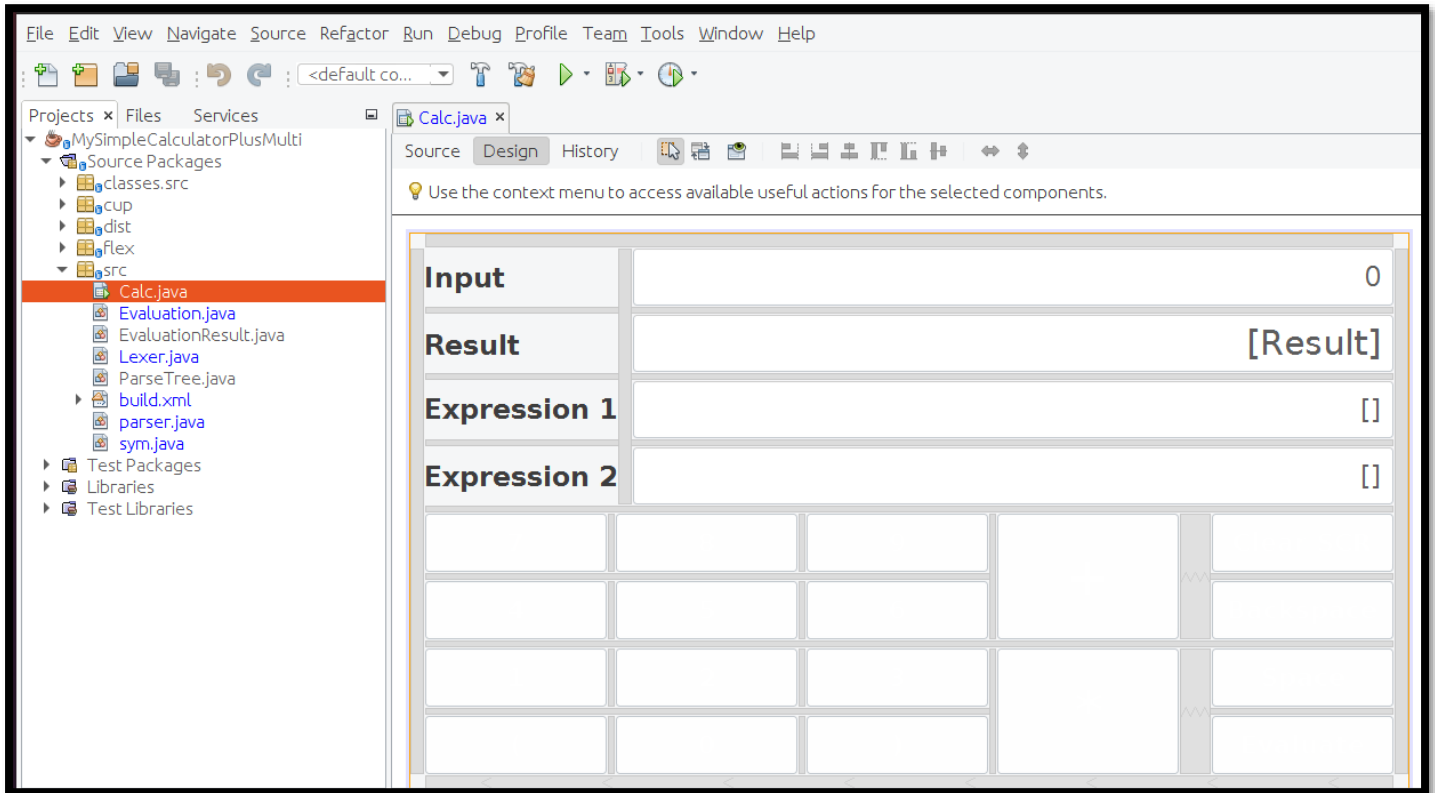
A standard tool for development of lexical analyzers is **Jflex** and we can see **lexical.flex** file where it consists of User code, Options and Translation rules

- ✓ **User code:** This is the place for *package* declaration and *import* statements.
- ✓ **Options:** customize the generated lexer, declare constants, variables, regular definitions
- ✓ **Translation rules:** have the form $p \{ action \}$

where p is a regular expression and action is Java code specifying what the lexer executes when a sequence of characters of the input string matches p .

This section contains regular expressions and actions, i.e., Java code, that will be executed when the scanner matches the associated regular expression.

Output Screen



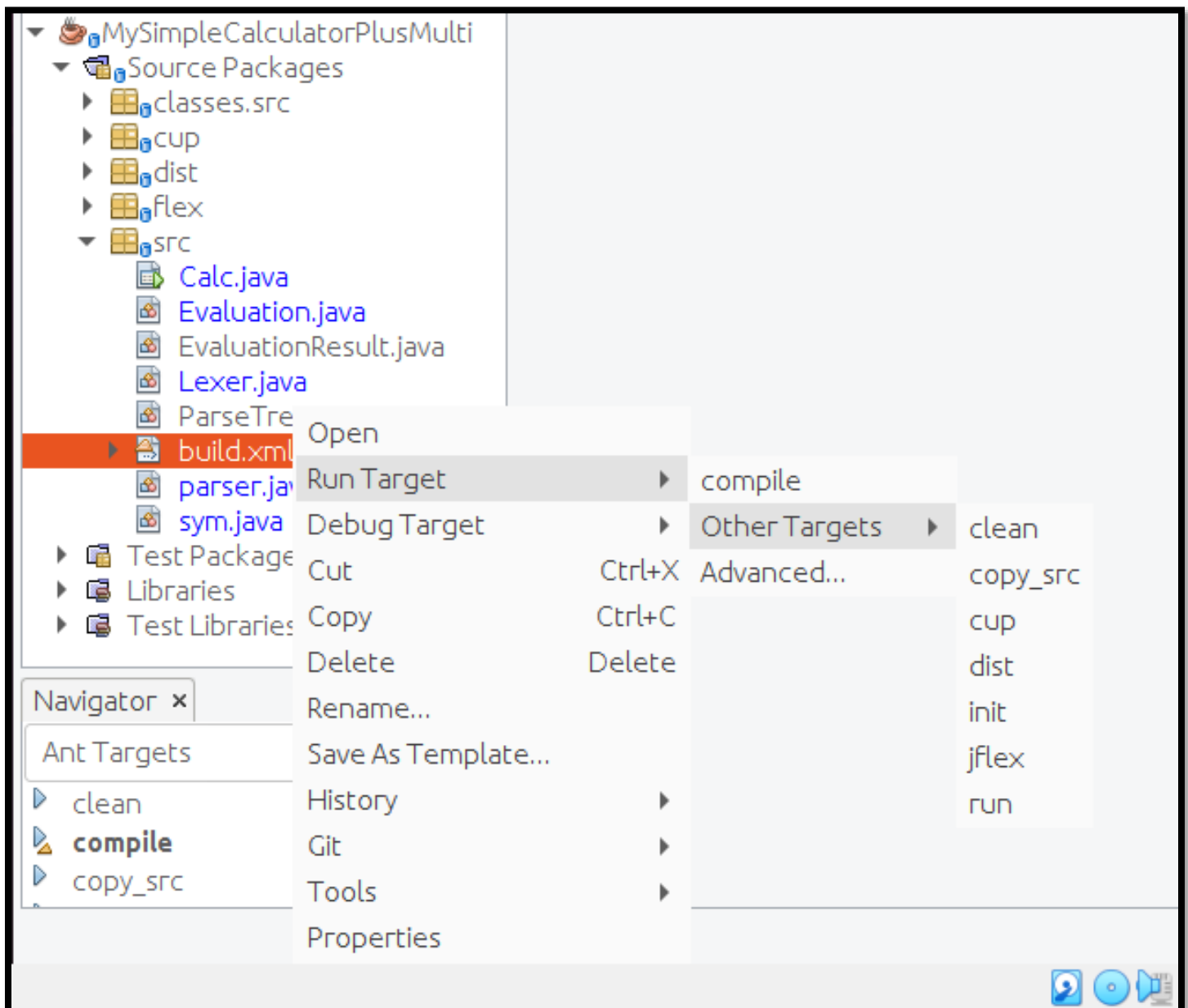
The above file and location show the interface design and where the output-oriented programs are performed.

The *Evaluation.java* and *EvaluationResult.java* link the calculator interface with operation functionality and displays the output on screen depending on input. The Calculator take integer input and compute the operation and display it. If the input is infix, then output should be value of operation and show the prefix and postfix expression of it.

ParseTree.java is having getter and setter for the parse tree with token, level, lexeme, and type.

- *parser.java* and *sym.java* are generated by CUP.
- *Lexer.java* is generated by Jflex.

To generate the above file by cup and Jflex, we use **build.xml** to run in different targets to compile and run. The detail is show in below figure.



With the bulid.xml we can clean and run target on other targets to get the required and make our file executable. The main targets are cup and jflex or we can have dist and init for compiling since we clean it in build.xml.

Therefore, we can run on all targets or we can change the clean configuration in bulid.xml and in addition we can also set the source of cup file and flex file.

Output after running the program (my calculator)

Calculator Screen

The image shows a Java Swing window titled "Calculator Evalutated by jFlex and Cup". Inside the window, there are four text input fields stacked vertically, each with a label to its left: "Input" (containing "0"), "Result" (containing "[Result]"), "Expression 1" (containing "[]"), and "Expression 2" (containing "[]"). Below these fields is a grid of buttons. The first three rows of buttons contain digits: 7, 8, 9; 4, 5, 6; and 1, 2, 3. The fourth row contains parentheses: "(", "0", and ")". To the right of the digit buttons are two larger operator buttons: "+" and "*". To the right of the operator buttons are four buttons: "Clear SCR" (red), "Backspace" (orange), "Space" (orange), and "Evaluate" (green).

Above figure is the main interface or page of the calculator where we can see four text field, operator buttons, operant buttons, and evaluation button.

Input field is for the user to input the expression such as infix notation or postfix notation or prefix notation. Depending on the input the next field, which is **Result field** will display the result or value of expression given in input. Then the next two field, **Expression 1 and 2 field** will display the expression (infix or postfix or prefix) which is two expression notations other than the input notation.

Additional buttons are there to clear screen, backspace, and space. Space button is used to input the expression such as postfix and prefix notation to have space between operator and operant.

To understand more how the calculator works, following figures will help you to make things clear.

Infix Input

Calculator Evaluated by jFlex and Cup

Input	6+3
Result	9.0
Expression 1	6 3 +<Postfix>
Expression 2	+ 6 3<Prefix>

7

8

9

+

Clear SCR

4

5

6

*

Backspace

1

2

3

Space

(

0

)

Evaluate

Simple input like infix (like in math) will display the value of expression and in different expressions (postfix and prefix expression).

Input with parentheses ‘ (‘ and ‘) ‘

Calculator Evaluated by jFlex and Cup

Input	(3+3)*2
Result	12.0
Expression 1	3 3 + 2 *<Postfix>
Expression 2	* + 3 3 2<Prefix>

7

8

9

+

Clear SCR

4

5

6

*

Backspace

1

2

3

Space

(

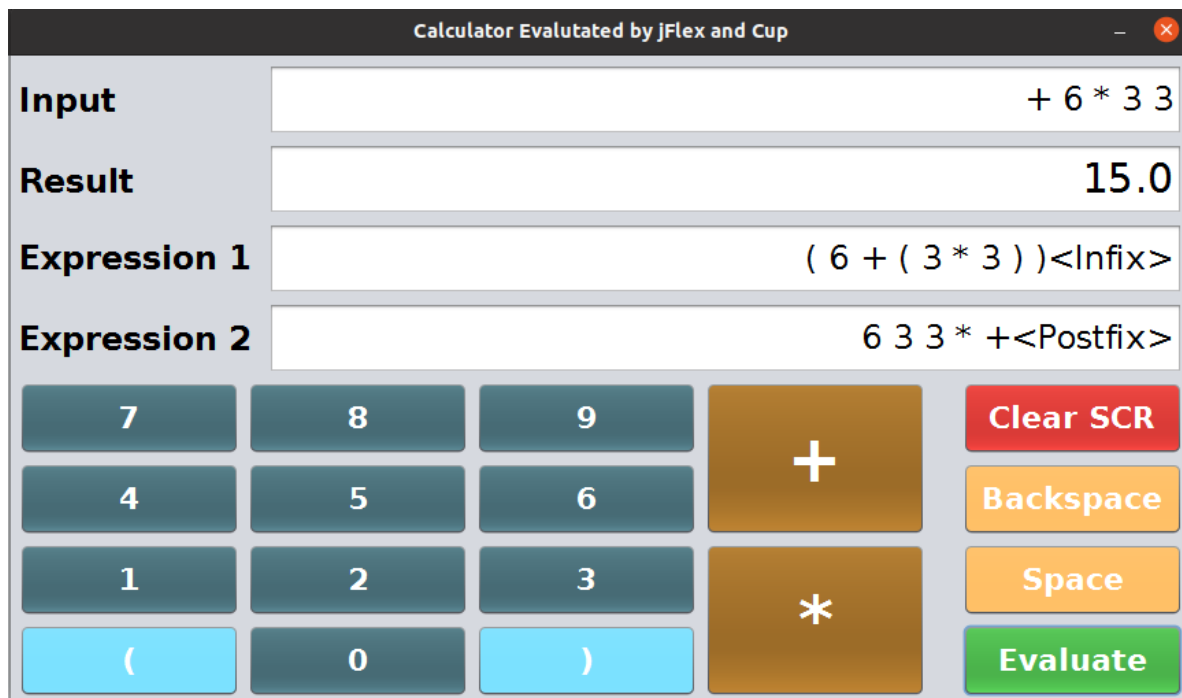
0

)

Evaluate

To show the priority of () expression and calculation with it.

Prefix Input



Calculator Evaluated by jFlex and Cup

Input + 6 * 3 3

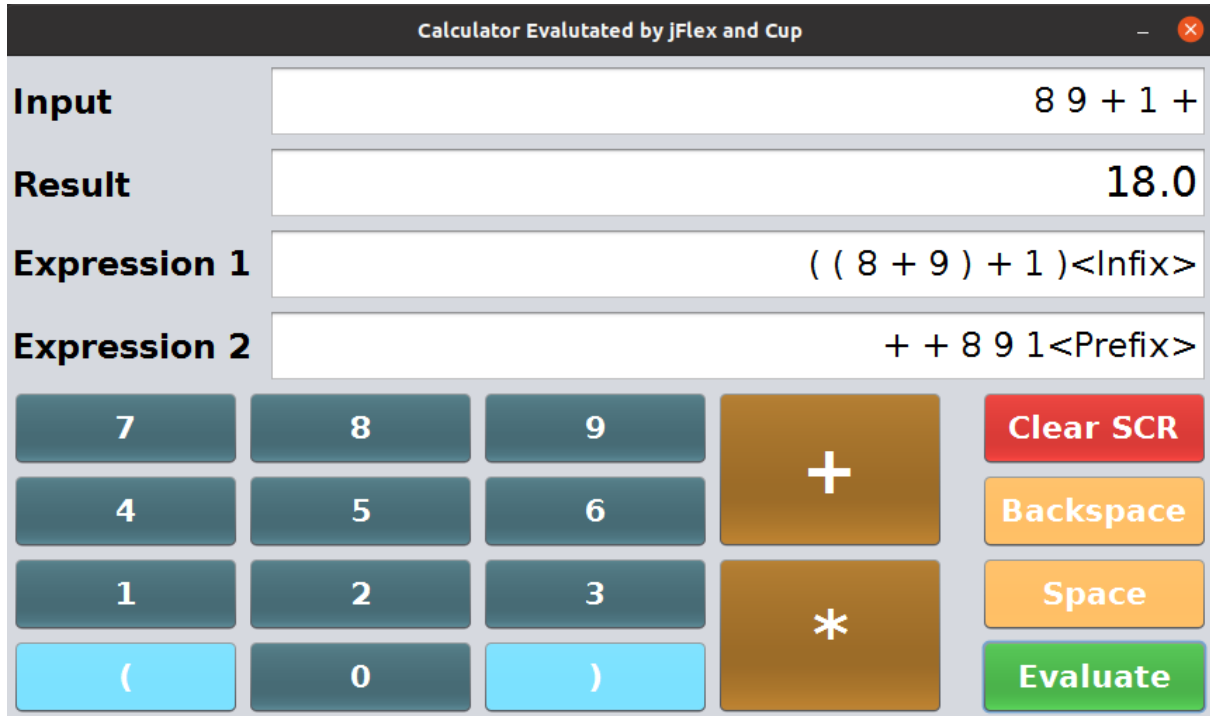
Result 15.0

Expression 1 (6 + (3 * 3))<Infix>

Expression 2 6 3 3 * +<Postfix>

Calculator buttons: 7, 8, 9, +, Clear SCR, 4, 5, 6, Backspace, 1, 2, 3, *, Space, (, 0,), Evaluate.

Postfix Input



Calculator Evaluated by jFlex and Cup

Input 8 9 + 1 +

Result 18.0

Expression 1 ((8 + 9) + 1)<Infix>

Expression 2 + + 8 9 1<Prefix>

Calculator buttons: 7, 8, 9, +, Clear SCR, 4, 5, 6, Backspace, 1, 2, 3, *, Space, (, 0,), Evaluate.

Above two figure shows that the expression 1 and 2 depends on the input expression. If the input is prefix expression, then output expression is of infix and postfix notation of the input expression.

Syntax error

Calculator Evaluated by jFlex and Cup

Input + 6 * 3

Result Can't recover from previous error(s)

Expression 1 Syntax error

Expression 2 Syntax error

7 8 9 + Clear SCR

4 5 6 Backspace

1 2 3 *

(0) Evaluate

If the input is wrong expression, then error message is displayed as shown in above figure. For example, if the input expression is not infix or postfix or prefix notation then the result and expression 1 & 2 will have error while we evaluate the input.

Conclusion

The key concepts used in this project was the execution of the grammar or rules in the cup. The parse tree records a sequence of rules the parser applies to recognize the input. In my grammar, the start symbol is "S". It is the root of the parse tree. Each interior node represents a non-terminal as represented in the grammar rule like statements, declarations (infix, postfix, and prefix), and operation. Each leaf node represents a token or a terminal symbol.

All the evaluation of the expressions and statements are recursively done. Error checking is also done where two operands are checked for their compatibility with operator and are shown if incompatible expressions are found.

The program is translated into Prefix Notation and Postfix Notation and displayed whenever the program is run.

Reference

Apache NetBeans. (2020). The Apache Software Foundation. Retrieved from

<https://netbeans.apache.org/download/index.html>

Jflex (2020, May 3). *What is it?* Retrieved from <https://jflex.de/>

Lexical Analysis using Jflex. (n.d). Retrieved from

<https://www.cs.auckland.ac.nz/courses/compsci330s1c/lectures/330ChaptersPDF/Chapt1.pdf>

Wielenga, G. (2014, February 26). Top 5 features of NetBeans 8. *Geertjan's Blog*. Retrieved from

<https://blogs.oracle.com/geertjan/top-5-features-of-netbeans-8>