Group 4:

- **Dilan Shaminda**
- **Sadaf Alvani**
- **Server Khalilov**
- **Tareq Nasif**

1. Overview

This repository consists of the Assignment 1 for the CLPT lecture. The main goal was to implement the methods of evaluation & pretty-printering the expressions in order to disambiguate them by putting brackets around.

The structure of the projects is based on the Java Cup (Parser generator), Java Flex (Scanner generator), Gradle (Build system for the JVM) and implementation of AST using Java Classes.

2. Task 1.

According to the provided example in the code, we have successfully added the operation **Subtraction** and **Division** in the exprs.cup file and in the exprs.flex file.

3. Task 2.

The binding strength has been solved using the **precedence** keyword for the .cup file, which specifies the precedence for all the defined operators. In our solution, it looks like the following piece of code:

precedence left PLUS, MINUS;
precedence left TIMES, DIV;
precedence left NEGATION

The order of the precedence is from the bottom – highest to the top – lowest. Therefore, it means that NEGATION has higher precedence than TIMES, DIV, which have the higher precedence than PLUS, MINUS accordingly. The keyword **left** state that we have defined the association from left to right. If we wanted to do it in the other way around, we could have used the keyword **right**.

4. Task 3.

The AST (Abstract Syntax Tree) is represented as the list of Java Classes (different types of expressions accordingly are describing itself). The Visitor Design pattern has an interface, which is called **ExprVisitor**. We called a corresponding **accept** method to get the values. We have implemented accept method in a recursive manner. It consists of a bunch of visitor methods, which are described for each type of expressions for the AST. The idea is to separate an algorithm for pretty-printing from an object structure on which it operates. It is a really useful way of adding the functionality without changing the classes itself. The same approach we have applied for the implementation of the evaluation of expression method, which could be seen in the interface **ExprEvalVisitor** and **ExprEvaluateVisitor**.